졸업논문청구논문

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

박 서 진 (Park, Seo Jin)

19039

과학영재학교 경기과학고등학교

2022

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

[논문제출 전 체크리스트]

1. 이 논문은 내가 직접 연구하고 작성한 것이다. ☑
2. 인용한 모든 자료(책, 논문, 인터넷자료 등)의 인용표시를 바르게 하였다. ☑
3. 인용한 자료의 표현이나 내용을 왜곡하지 않았다. ☑
4. 정확한 출처제시 없이 다른 사람의 글이나 아이디어를 가져오지 않았다. ☑
5. 논문 작성 중 도표나 데이터를 조작(위조 혹은 변조)하지 않았다. ☑
6. 다른 친구와 같은 내용의 논문을 제출하지 않았다. ☑

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

Advisor : Teacher Park, Kiehyun

by

19039 **Park, Seo Jin**

Gyeonggi Science High School for the gifted

A thesis submitted to the Gyeonggi Science High School in partial fulfillment of the requirements for the graduation. The study was conducted in accordance with Code of Research Ethics.*

2021. 8. 3.

Approved by
Teacher Park, Kiehyun
[Thesis Advisor]

*Declaration of Ethical Conduct in Research: I, as a graduate student of GSHS, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

박 서 진

위 논문은 과학영재학교 경기과학고등학교 졸업논문으로
졸업논문심사위원회에서 심사 통과하였음.

2021년 8월 3일

심사위원장 김 학 성 (인)

심사위원 이 호 (인)

심사위원 박 기 현 (인)

Estimation of Sea Surface Temperature around the Korean Peninsula Using NOAA/AVHRR Data

Abstract

Put your abstract here. It is completely consistent with 한글초록.

NOAA/AVHRR 자료를 이용한 한반도 주변 해역의 해수면 온도 산출

초 록

초록(요약문)은 가장 마지막에 작성한다. 연구한 내용, 즉 본론부터 요약한다. 서론 요약은 하지 않는다. 대개 첫 문장은 연구 주제 (+방법을 핵심적으로 나타낼 수 있는 문구: 실험적으로, 이론적으로, 시뮬레이션을 통해)를 쓴다. 다음으로 연구 방법을 요약한다. 선행 연구들과 구별되는 특징을 중심으로 쓴다. 뚜렷한 특징이 없다면 연구방법은 안써도 상관없다. 다음으로 연구 결과를 쓴다. 연구 결과는 추론을 담지 않고, 객관적으로 서술한다. 마지막으로 결론을 쓴다. 이 연구를 통해 주장하고자 하는 바를 간략히 쓴다. 요약문 전체에서 연구 결과와 결론이 차지하는 비율이 절반이 넘도록 한다. 읽는 이가 요약문으로부터 얻으려는 정보는 연구 결과와 결론이기 때문이다. 연구 결과만 레포트하는 논문인 경우, 결론을 쓰지 않는 경우도 있다.

Contents

Abstract i

초록 ii

Contents iii

List of Tables v

List of Figures vi

I **서론** 1

I.1 연구의 필요성 및 목적 1

I.2 이론적 배경 2

I.2.1 기상 위성 2

I.2.2 NOAA 위성 2

I.2.3 Terra/Aqua 위성 3

I.2.4 인공위성 자료 4

I.2.5 SST 산출 알고리즘 4

II **연구 방법 및 과정** 7

II.1 데이터 파악 7

II.2 연구에 사용한 데이터 9

II.3 자료 처리 11

II.4 데이터 파악 및 수집 11

III **연구 결과** 12

III.1 자료 처리 12

III.1.1 SST를 히스토그램 12

III.1.2 SST를 지도에 표출 13

III.2 레벨3 자료 산출 14
III.2.1 일평균값 산출 14
III.2.2 주평균값 산출 15
III.2.3 월평균값 산출 16
III.2.4 해수면 온도 변동 17
IV 결론 **18**
V 부록 **19**
V.1 MODIS_hdf_utilities.py 19
V.2 1.daily_classify_using_AVHRR_asc_SST.py 39
V.3 2.statistics_AVHRR_asc_SST_alldata_and_creating_NCfile.py 45
V.4 4.draw_HIST_and_MAP_statistics_AVHRR_asc_SST_NCfile.py 50
References **53**

List of Tables

Table 1. Description for AVHRR channels Channel. 3
Table 2. Description for MODIS channels. 5
Table 3. 해양위성센터에서 다운로드 가능한 SST 데이터. 7
Table 4. 사용한 NOAA/AVHRR SST data. 10

List of Figures

Figure 1. NOAA/AVHRR SST 자료 텍스트 파일 캡처 화면. 8

Figure 2. KOSC에서 배포한 NOAA/AVHRR SST 자료. 9

Figure 3. SST의 히스토그램(NOAA/AVHRR). 12

Figure 4. (a) KOSC에서 배포한 NOAA/AVHRR SST. (b) 직접 그린 NOAA/AVHRR SST. 13

Figure 5. SST 일평균값 산출 결과(NOAA/AVHRR). 14

Figure 6. SST 주평균값 산출 결과(NOAA/AVHRR). 15

Figure 7. SST 월평균값 산출 결과(NOAA/AVHRR). 16

Figure 8. Variation of monthly mean values of SST from 2010 to 2020. 17

I. 서론

I.1 연구의 필요성 및 목적

복잡하게 구성된 지구의 순환 체계에서 Sea Surface Temperature (SST)는 빠뜨릴 수 없는 요소이다. 기후에 밀접하게 영향을 주고받는 SST는 몇몇 해역에서 대기에 강제력을 행사하고, 다른 해역에서는 대기에 영향을 받으며 억지력으로서 작용한다. 계절에 따라 SST와 대기가 미치는 영향의 비중이 달라지는 해역도 존재한다 [1].

SST는 태풍이나 집중호우 등의 위험기상의 발생가능성 또한 SST의 변동성과 연관지어 예측할 수 있는 만큼 SST를 관측하고 그 경향성을 파악하는 것은 지구 환경을 이해하는 데에 굉장히 중요하다 [2].

SST를 관측하는 방법으로는 크게 해양 부이를 이용한 관측과 인공위성 자료를 통한 산출법이 있다. 전자의 경우 구름과 같은 오차 원인을 배제하고 직접적으로 정확한 데이터를 얻을 수 있다는 장점이 있으나, 부이가 위치하는 한 점의 값만을 얻을 수 있기 때문에 폭넓은 지역의 해수면 온도를 알 수 없다는 단점이 있다. 그와는 반대로 인공위성 자료를 통한 산출법은 대기와 다른 여러 요인들로 인한 오차를 계산해야 하나, 위성으로 관측할 수 있는 광범위한 해역의 정보를 알 수 있다는 것이 장점이다.

본 연구에서는 인공위성 자료를 이용하여 한반도 주변 해역의 SST를 산출해 보고자 한다.

I.2 이론적 배경

I.2.1 기상 위성

기상위성이란 지구의 기상현상과 대기를 관측하기 위한 목적의 인공위성들의 분류이며, 우리가 현재 사용하는 기상위성은 궤도에 따라 정지궤도위성과 극궤도위성으로 나뉜다.

정지궤도위성은 적도 상공에 위치해, 약 35,800 km 높이에서 지구와 같은 각속도로 지구 주위를 공전하기 때문에 지상의 관측자가 보았을 때에는 하늘에 고정된 것처럼 느껴지므로 이와 같은 명칭이 붙었다. 정지궤도위성은 지구의 약 $\frac{1}{4}$ 정도 되는 고정된 면적을 관측할 수 있으며 이 때문에 한 지역의 연속적인 기상 상태 변화 등을 관찰하는 데에 있어 유용하다.

극궤도위성은 남극과 북극을 통과하여 지구 주위를 공전하는 위성으로, 고도는 약 800 – 1,500 km 정도이다. 이는 하루에 전체 지구를 약 2회 관측할 수 있으며, 고도가 기상위성에 비해 낮아 세기가 약한 파장도 인식할 수 있으며, 극지의 얼음, 해양, 에너지의 순환 등 다양한 현상을 관측할 수 있다.

I.2.2 NOAA 위성

National Oceanic and Atmospheric Administration (NOAA)에서 진행하는 Polar Operational Environmental Satellite (POES) 프로젝트의 일부로 NOAA 위성을 운용하고 있다. 이 위성은 직하점을 중심으로 55.4° 안쪽의 범위를 주사할 수 있다. 탑재되어 있는 주 관측 센서는 Advanced Very High Resolution Radiometer (AVHRR)와 Television InfraRed Observation Satellite Operational Vertical Sounder (TOVS) 등이 있다. 이 가운데 AVHRR은 5개의 채널을 가졌으며 각각의 파장과 주 용도는 Table 1과 같다.

Table 1. Description for AVHRR channels Channel.

Channel Number	Wavelength (μm)	Typical Use
1	0.58 ~0.68	Daytime cloud and surface mapping
2	0.725 ~1.00	Land-water boundaries
3a	1.58 ~1.64	Snow and ice detection
3b	3.55 ~3.93	Night cloud mapping, Sea surface temperature
4	10.30 ~11.30	Night cloud mapping, Sea surface temperature
5	11.50 ~12.50	Sea surface temperature

I.2.3 Terra/Aqua 위성

1999년 12월 18일 발사되어 2000년 2월 24일 부터 자료를 송신한 Terra (EOS AM-1) 위성은 하루에 한 지점을 2번 관측하는 극궤도위성이다. 지구 환경과 기후의 변화를 관측하는 것이 목표인 이 위성은 Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), Clouds and the Earth's Radiant Energy System (CERES), Multi-angle Imaging SpectroRadiometer (MISR), Moderate-resolution Imaging Spectroradiometer (MODIS), Measurements of Pollution in the Troposphere (MOPITT) 로 총 6 가지의 센서들을 탑재하였다.

Aqua 위성은 2002년 5월 4일 지표면과 대기 중의 물에 관한 연구를 위하여 발사되었으며, Atmospheric Infrared Sounder (AIRS), the Advanced Microwave Sounding Unit (AMSU-A), the Humidity Sounder for Brazil (HSB), the Advanced Microwave Scanning Radiometer for EOS (AMSR-E), the Moderate-Resolution Imaging Spectroradiometer (MODIS), and the Clouds and the Earth's Radiant Energy System (CERES) 로 총 6가지 센서들을 탑재하였으나, 그중 AMSR-E와 HSB가 손상되어 작동을 멈추었고, AMSU-A와 CERES는 일부 고장이 발생하였으나 여전히 작동하고 있다. Terra와 Aqua 위성은 Aura 위성과 함께 Earth Observing System(EOS)의 일부이다.

MODIS는 Terra와 Aqua 위성의 핵심 탑재체이다. 크기 1.0 m × 1.6 m × 1.0 m, 질량 228.7 kg의 MODIS는 위성에 탑재되어 705 km의 고도에서 55°의 시야각, 2330 km의 관측폭으로 하루 한 번 혹은 두 번 같은 지점을 관측한다. 총 36 개인 각 채널의 해상도는 각각 250 m(채널 1 – 2), 500 m(채널 3 – 7), 1 km(채널 8 – 36)이며 그 중 SST 관측에 쓰이는 것은 약 3.7 – 4.1 μm의 대역폭을 가지고 있는 20, 21, 22, 23 번 채널과 10.8 – 12.3 μm의 31, 32 번 채널이다. 자세한 정보는 Table 2에 나타내었다.

I.2.4 인공위성 자료

인공위성 자료는 처리 정도에 따라 레벨 0, 레벨 1A, 레벨 1B, 레벨 2, 레벨 3, 레벨 4 데이터로 나뉜다 [3].

레벨 0 데이터는 우주선에서 지상으로 전송하는 데 쓰이는 통신 정보만을 제거한 상태의 페이로드 데이터를 의미하며, 레벨 1A 데이터는 시간을 참조하여 레벨 0 데이터를 재구성하고 기하적 보정 등 보조 자료를 주석으로 추가한 상태이다. 레벨 1B 데이터는 그것에서 센서의 특성과 복사량에 대한 보정이 이루어진 결과물로, 이 단계부터는 센서 보정이 변경된다면 다른 데이터로 대체되어야만 한다.

레벨 2 데이터는 이들을 이용하여 지구물리학적으로 의미있는 변수들을 도출하여 SST(Sea Surface Temperature), OC(Ocean Color) 등의 그룹으로 분류한 것이고, 레벨 3 데이터는 그러한 데이터를 일정 기간 동안 일정 구역 집계한 기록이다.

마지막으로 레벨 4 데이터는 하위 레벨 데이터에 대한 분석을 말한다.

본 연구에서는 인공위성을 이용한 SST 산출 방식을 채택하여 NOAA 위성의 AVHRR 센서로 관측한 레벨 2 데이터를 레벨 3 데이터로 가공하여 분석하는 것이 목적이다.

I.2.5 SST 산출 알고리즘

인공위성 자료를 통해 SST 데이터를 산출하는 데에는 MCSST(Multi-Channel Sea Surface Temperature)와 CPSST(Cross Product Sea Surface Temperature) 등 여러 기법이 존재한

Table 2. Description for MODIS channels.

Primary Use	Band	Bandwidth1	Spectral Radiance	Required SNR
Land/Cloud/Aerosols Boundaries	1	620 – 670	21.8	128
	2	841 – 876	24.7	201
Land/Cloud/Aerosols Properties	3	459 – 479	35.3	243
	4	545 – 565	29.0	228
	5	1230 – 1250	5.4	74
	6	1628 – 1652	7.3	275
	7	2105 – 2155	1.0	110
Ocean Color/ Phytoplankton/ Biogeochemistry	8	405 – 420	44.9	880
	9	438 – 448	41.9	838
	10	483 – 493	32.1	802
	11	526 – 536	27.9	754
	12	546 – 556	21.0	750
	13	662 – 672	9.5	910
	14	673 – 683	8.7	1087
	15	743 – 753	10.2	586
	16	862 – 877	6.2	516
Atmospheric Water Vapor	17	890 – 920	10.0	167
	18	931 – 941	3.6	57
	19	915 – 965	15.0	250
Surface/Cloud Temperature	20	3.660 – 3.840	0.45(300K)	0.05
	21	3.929 – 3.989	2.38(335K)	0.20
	22	3.929 – 3.989	0.67(300K)	0.07
	23	4.020 – 4.080	0.79(300K)	0.07
Atmospheric Temperature	24	4.433 – 4.498	0.17(250K)	0.25
	25	4.482 – 4.549	0.59(275K)	0.25
Cirrus Clouds Water Vapor	26	1.360 – 1.390	6.00	150(SNR)
	27	6.535 – 6.895	1.16(240K)	0.25
	28	7.175 – 7.475	2.18(250K)	0.25
Cloud Properties	29	8.400 – 8.700	9.58(300K)	0.05
Ozone	30	9.580 – 9.880	3.69(250K)	0.25
Surface/Cloud Temperature	31	10.780 – 11.280	9.55(300K)	0.05
	32	11.770 – 12.270	8.94(300K)	0.05
Cloud Top Altitude	33	13.185 – 13.485	4.52(260K)	0.25
	34	13.485 – 13.785	3.76(250K)	0.25
	35	13.785 – 14.085	3.11(240K)	0.25
	36	14.085 – 14.385	2.08(220K)	0.35

다. (박경혜, 정종률, 최병호, 김구, 1994) SST 산출에 쓰이는 채널은 22, 23번(단파)와 31, 32번(장파)이며, 각각의 채널에서는 지표면을 흑체로 가정하고 슈테판-볼츠만 법칙을 이용하여 밝기온도를 구한다. McMillin과 Crosby(1984)의 연구 결과에 의하면 수증기흡수계수 ki, kj에 대하여 =kjkj-ki일 때, SST=Tj+(Ti-Tj)의 값을 가진다.

그렇게 도출한 단일채널 SST의 값을 이용하여 아래와 같은 총 세 가지 기법으로 MCSST를 산출한다 [4].

MCSST(3, 4)=T11+1.616(T3.7-T11)+1.07 (dual window) MCSST(4, 5)=T12+3.15(T11-T12)+0.10 (split window) MCSST(3, 4, 5)=T11+0.943(T3.7-T12)+0.61 (triple window)

MCSST를 구하는 식에서는 수증기의 적외선 흡수율이 상수라고 가정하나, 실제로는 온도와 관계 있는 비선형적 함수로서 나타나고, 이에 따라 건조한 극지방이나 고온의 지역에서 산출한 결과와는 오차가 발생하게 된다. 따라서 이를 보완하기 위하여 개발된 비선형 알고리즘이 CPSST이다. (Walton et al, 1998)

II. 연구 방법 및 과정

≪≪≪〈 HEAD

II.1 데이터 파악

해양위성센터에서 제공하는 SST 데이터를 다운받을 수 있는 경로를 확인하였다. 접근할 수 있는 데이터는 2020년 4월 29일 기준으로 Table 3과 같다.

Table 3. 해양위성센터에서 다운로드 가능한 SST 데이터.

센서명	자료시작시기	자료종료시기 (2020. 4. 29. 기준)
AVHRR	2011. 9. 1.	2020. 4. 21.
MODIS (Aqua)	2011. 9. 1.	2020. 4. 6.
MODIS (Terra)	2011. 9. 1.	2020. 4. 7.
VIIRS	2016. 6. 17.	2020. 4. 27.

NOAA/AVHRR 자료는 Fig. 1과 같이 텍스트 파일의 형태로 배포되고 있다는 것을 알 수 있었다. 총 4개의 열로 저장되어 있으며, 첫번째 열부터 각각 인덱스, 위도, 경도, SST 임을 알 수 있는데, 자료가 산출되지 않은 경우에 ***로 표시되어 있다.

Terra/Aqua 위성의 MODIS를 구한 SST 자료는 HDF(Hierarchical Data Format) 형태로 배포되었다. HDF는 이름 그대로 계층적으로 구조화된 다차원 배열 데이터를 저장하기 위하여 HDF Gruop(https://www.hdfgroup.org/)에 의해 만들어진 파일 형식이다.

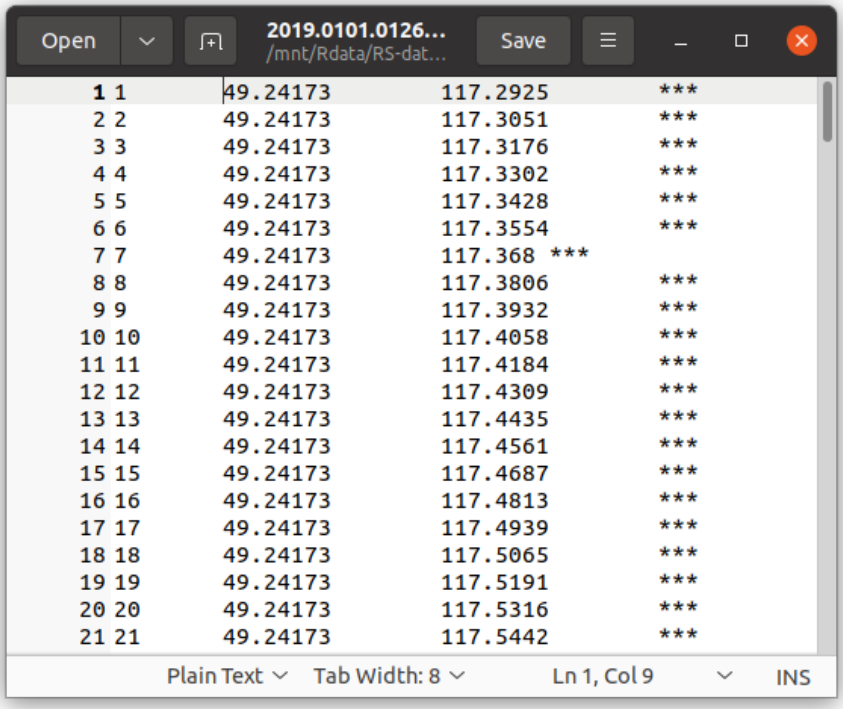

Figure 1. NOAA/AVHRR SST 자료 텍스트 파일 캡처 화면.

II.2 연구에 사용한 데이터

해양위성센터에서 배포한 MODIS의 SST 데이터는 구름이 제거되지 않아서 SST 값에 심각한 오류를 포함하고 있어 사용하지 않고, NOAA/AVHRR의 SST 레벨2 자료를 이용하여 연구를 진행하였다.

NOAA/AVHRR의 SST 레벨2 자료는 앞서 언급한 것 처럼 텍스트 파일 형태로 제공되고 있고, Figure 2와 같이 지도 위에 표출된 자료도 함께 제공되고 있다.

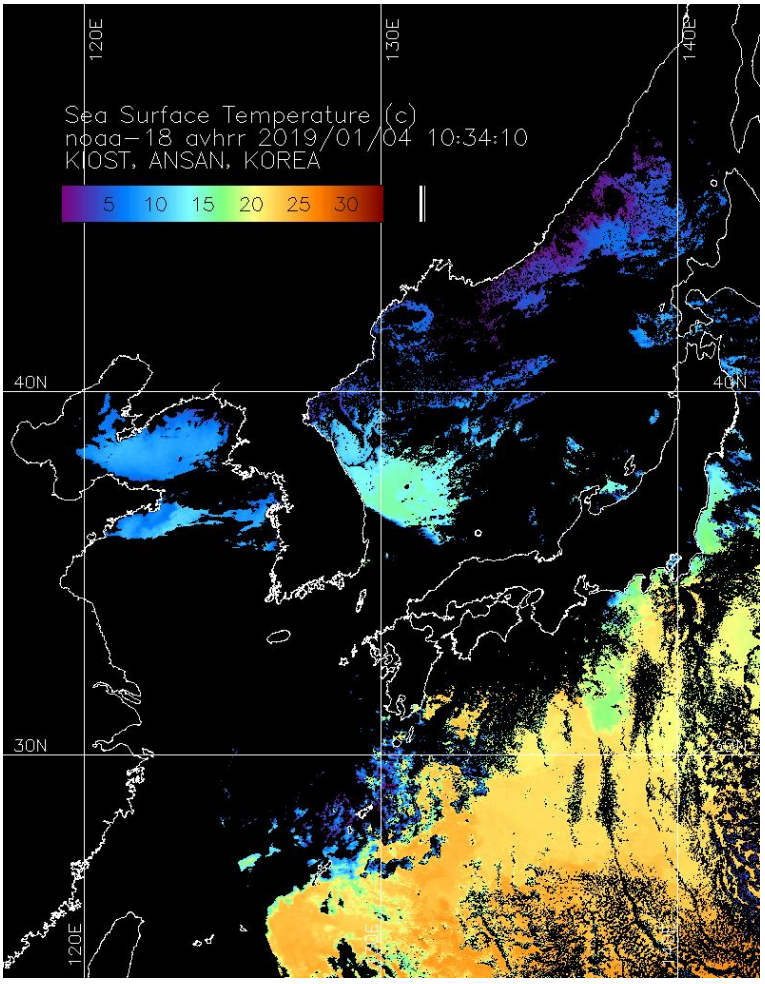

Figure 2. KOSC에서 배포한 NOAA/AVHRR SST 자료.

KOSC로 부터 다운받아 본 연구에 사용한 NOAA/AVHRR의 SST 자료의 정보는 Table

4와 같다.

Table 4. 사용한 NOAA/AVHRR SST data.

year	NOAA-15	NOAA-16	NOAA-17	NOAA-18	NOAA-19
2011	0	406	18	412	413
2012	12	1,184	11	1,073	1,141
2013	0	1,229	0	1,085	1,145
2014	0	533	0	1,056	728
2015	0	0	0	1,106	452
2016	0	0	0	1,022	1,177
2017	0	0	0	818	1,072
2018	0	0	0	912	937
2019	0	0	0	847	843
2020	0	0	0	526	527

II.3 자료 처리

NOAA/AVHRR의 SST 레벨2 자료를 위도, 경도 구간을 나눈 후, 일평균값, 주평균값, 월평균값을 산출하여 레벨3 자료를 만들었다. 자료 처리는 Phthon을 이용하여 실시하였다.

=======

II.4 데이터 파악 및 수집

해양위성센터에서 제공하는 SST 데이터를 다운받을 수 있는 경로를 확인하였다. 접근할 수 있는 데이터는 2020년 4월 29일 기준으로 Table 과 같다.

2012년부터 2019년까지의 8년 동안 Terra/Aqua 위성이 MODIS를 통해 수집한 자료를 연구에 이용하기로 결정하고, Github에서 다운로드한 웹 크롤링 파일을 이용하여 해양위성센터의 SST 데이터를 크롤링하는 table 과 같이 코드를 작성하였다. 2021년 6월 현재는 천리안위성 2호가 서비스를 시작하면서 사이트가 개편되어 데이터 배포 방식이 바뀌었어 아래의 코드가 실행되지 않을 수도 있다.

온라인 원격수업 환경에서 파일을 다운로드받기 위해 Chrome Remote Desktop을 이용하여 개인 노트북을 Ubuntu 운영체제의 서버 컴퓨터와 연결하여 사용할 수 있도록 하였으며, Ubuntu 프롬프트 명령어를 사용하여 파일 디렉토리를 탐색하는 방법을 학습하였다. 오랜 시간 동안 많은 양의 데이터를 다운받아야 하기 때문에 도중에 프롬프트 창을 닫더라도 계속 다운받을 수 있도록 nohup 명령어를 이용하여 백그라운드로 파일을 실행하였다. ≫≫≫〉 c2fb6d045ef38751fc9fe31d0bd3b77d1b59076f

III. 연구 결과

III.1 자료 처리

III.1.1 SST를 히스토그램

Python으로 코딩하여 SST Figure 3과 같이 자료의 히스토그램을 그려 자료의 신뢰도를 확인하였다.

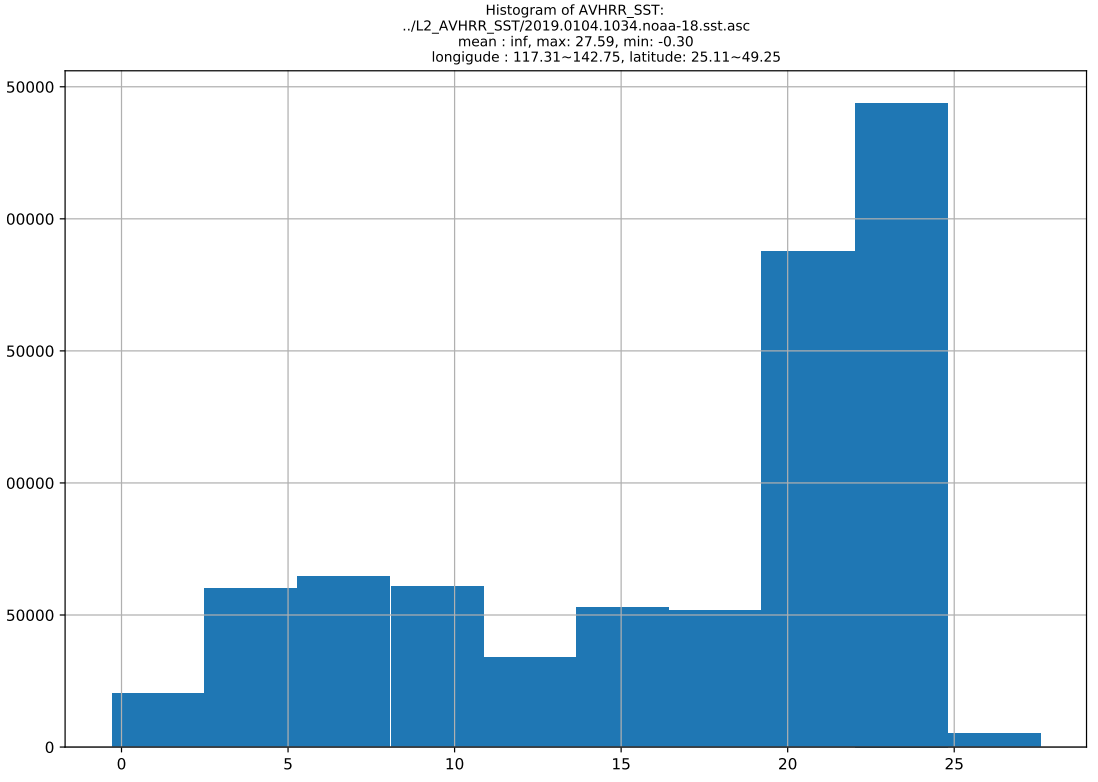

Figure 3. SST의 히스토그램(NOAA/AVHRR).

III.1.2 SST를 지도에 표출

Python으로 코딩하여 SST 자료를 지도 위에 표출하였다. KOSC에서 배포한 자료와 색깔은 다르게 표출하였으나, 원하는 모양대로 자료를 표출할 수 있었다.

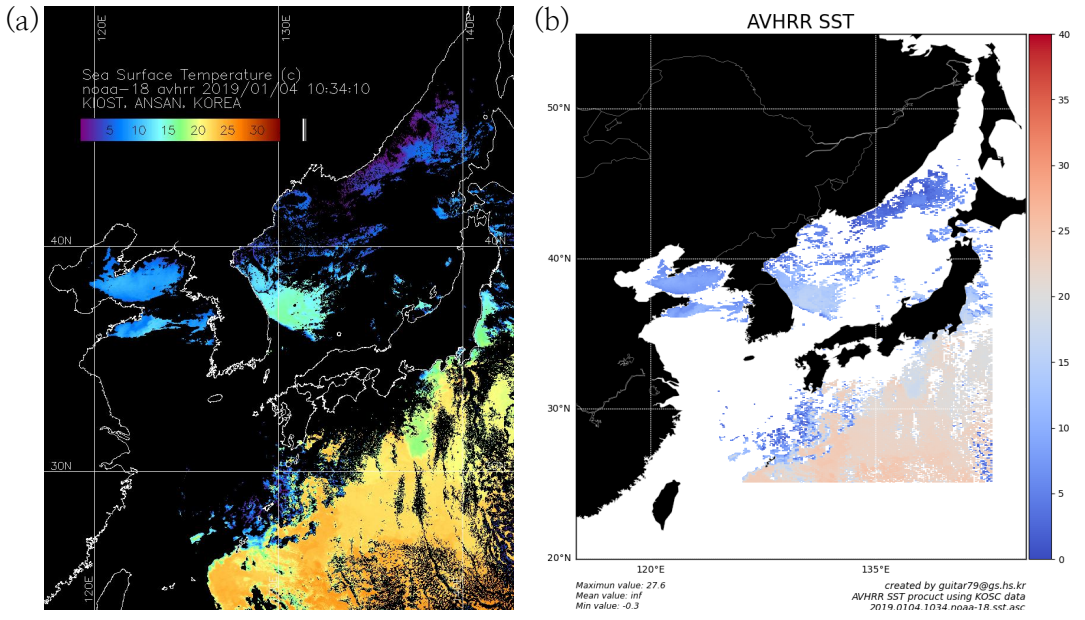

Figure 4. (a) KOSC에서 배포한 NOAA/AVHRR SST. (b) 직접 그린 NOAA/AVHRR SST.

III.2 레벨3 자료 산출

III.2.1 일평균값 산출

일평균값의 레벨3 자료를 산출하여 Figure 5와 같이 지도 위에 표출해 보았다.

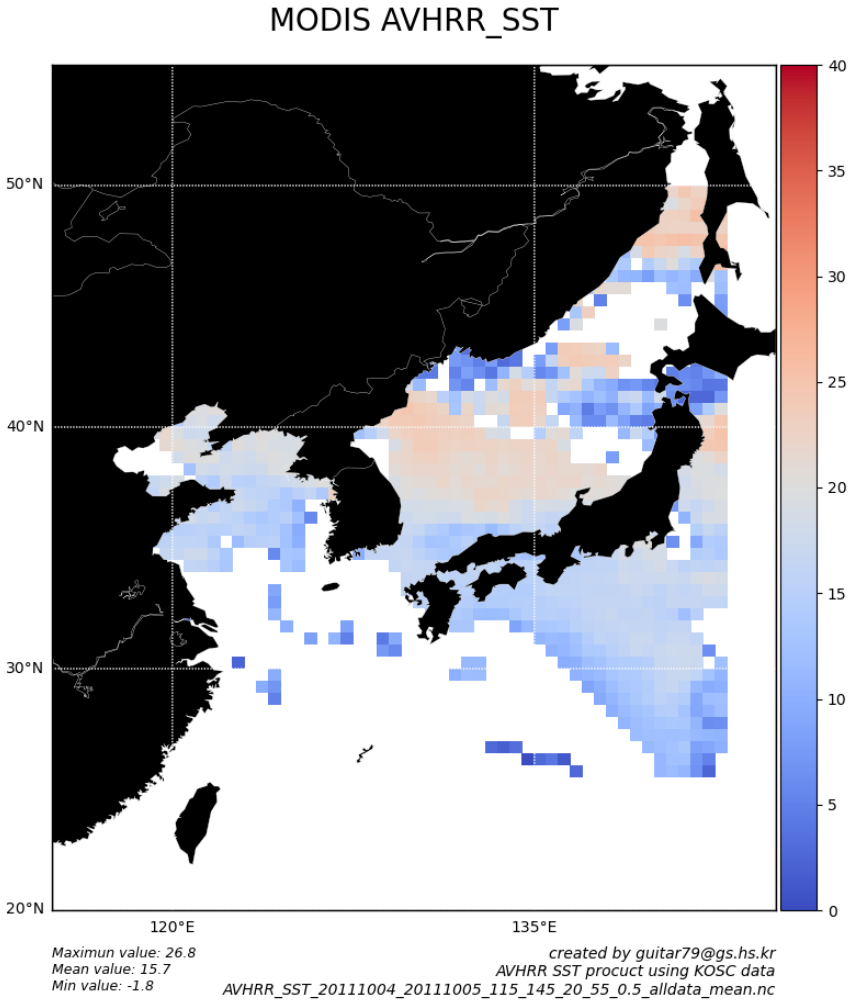

Figure 5. SST 일평균값 산출 결과(NOAA/AVHRR).

III.2.2 주평균값 산출

주평균값의 레벨3 자료를 산출하여 Figure 6과 같이 지도 위에 표출해 보았다.

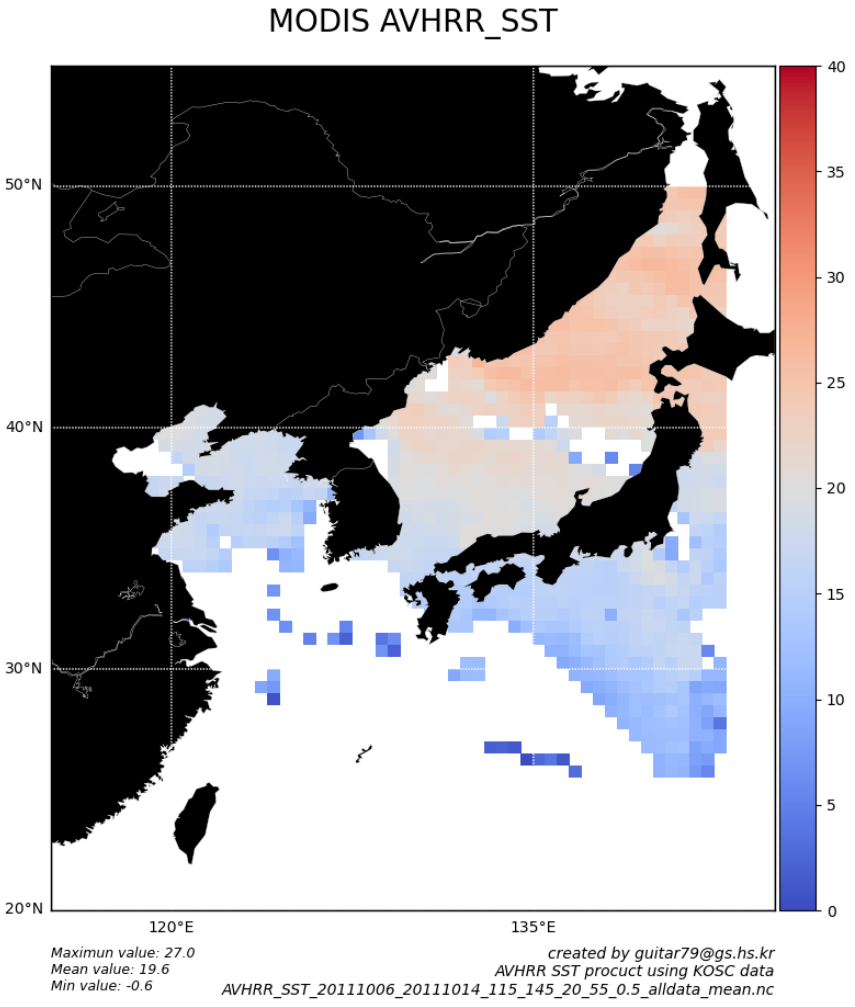

Figure 6. SST 주평균값 산출 결과(NOAA/AVHRR).

III.2.3 월평균값 산출

월평균값의 레벨3 자료를 산출하여 Figure 7과 같이 지도 위에 표출해 보았다.

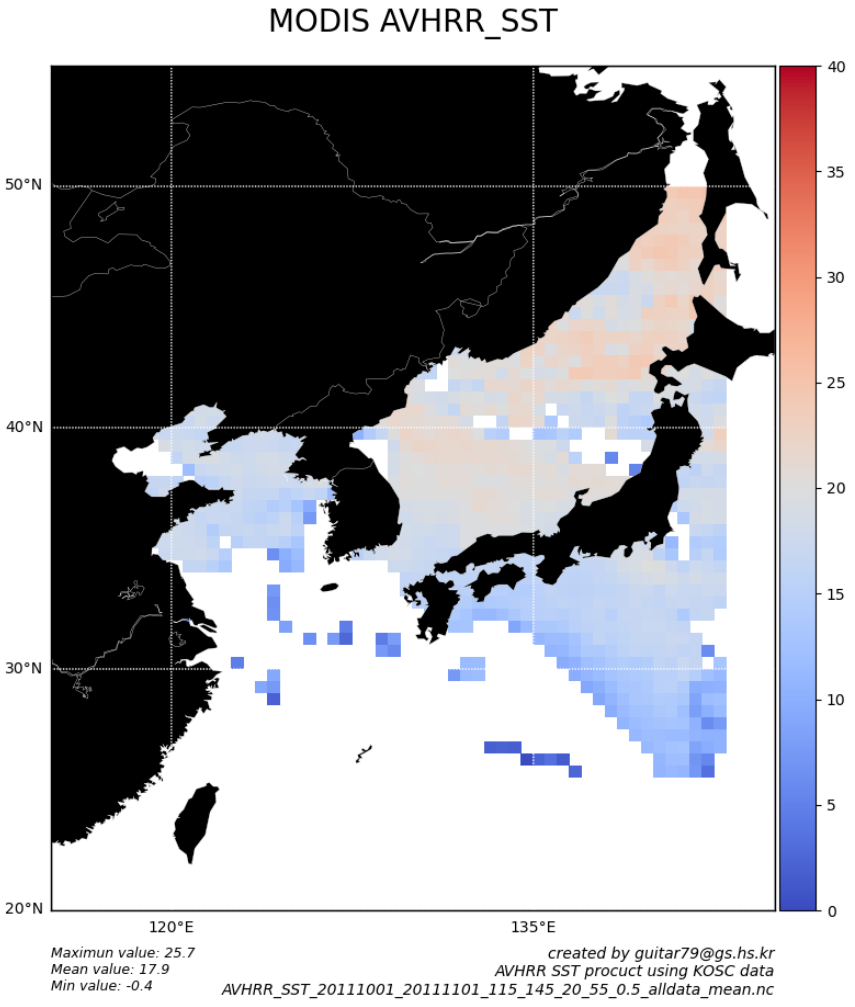

Figure 7. SST 월평균값 산출 결과(NOAA/AVHRR).

III.2.4 해수면 온도 변동

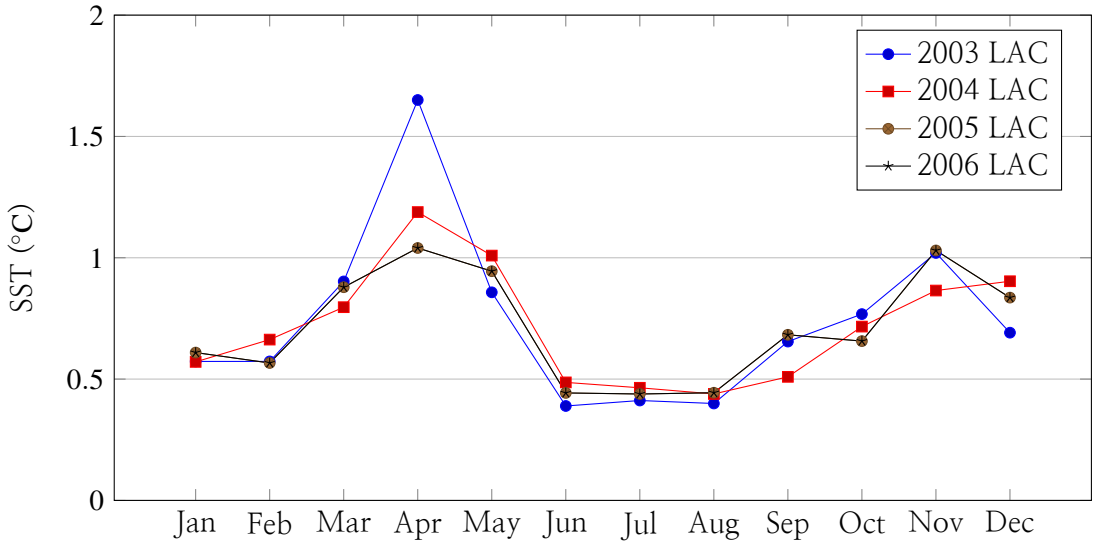

Figure 8. Variation of monthly mean values of SST from 2010 to 2020.

IV. 결론

글이라는 것은 개인의 개성이 담겨 있기 때문에 모든 사람들이 동일한 방식으로 표현하는 것은 아니다. 그러나 고대로부터 개인의 연구 내용을 글로써 타인에게 전달할 때, 효율적인 방법이라고 공감대를 형성하며 다듬어져 온 것이 지금의 논문 형태이다. 그러므로 처음 논문을 작성하는 학생들은 이 문서에서 지시하는 논문 작성 방식을 따르는 것을 권한다. 하지만 여기서는 다양한 논문들에 대해 일일이 사례를 들어 올바른 논문 작성법을 설명하기에는 한계가 있기에 간략하게만 소개를 했다. 여기서 설명되지 않은 부분들은 다른 사람들의 논문을 참고하자. 이미 서론을 작성하면서 많은 선행 연구 논문들을 읽어 봤을 것이다. 그 논문들에서는 데이터를 어떤 방식으로 표현하는지, 서론은 어떤 흐름으로 구성하는지 등을 살펴보자. 논문을 잘 쓰는 비결의 첫 번째는 논문을 많이 읽어 보는 것이다.

+ 첨언을 하자면, 본교의 영어논문작성법 수업에 사용되는 'Science Research Writing for Non-Native Speakers of English' 를 참고하면 많은 도움이 될 것이다.

V. 부록

자료 처리에 사용한 Python 코드를 부록으로 나타내었다.

모든 코드는 깃헙(https://github.com/guitar79/MODIS_hdf_Python.git)에 공유되어 있다.

V.1 MODIS_hdf_utilities.py

```
'''
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
Created on Sat Nov  3 20:34:47 2018
@author: guitar79
created by Kevin
#Open hdf file
NameError: name 'SD' is not defined
conda install -c conda-forge pyhdf
'''

from glob import glob
import numpy as np
from datetime import datetime

import os
from pyhdf.SD import SD, SDC

def write_log2(log_file, log_str):
    import os
    with open(log_file, 'a') as log_f:
        log_f.write("{}, {}\n".format(os.path.basename(__file__), log_str))
    return print ("{}, {}\n".format(os.path.basename(__file__), log_str))

def write_log(log_file, log_str):
    import time
    timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
    msg = '[' + timestamp + '] ' + log_str
    print (msg)
    with open(log_file, 'a') as f:
        f.write(msg + '\n')

#for checking time
cht_start_time = datetime.now()

```

```
#JulianDate_to_date(2018, 131) -- '20180511'
def JulianDate_to_date(y, jd):
    import calendar
    ################################################################
    #JulianDate_to_date(2018, 131) -- '20180511'
    #
    month = 1
    while jd - calendar.monthrange(y, month)[1] > 0 and month <= 12:
        jd = jd - calendar.monthrange(y, month)[1]
        month += 1
    #return datetime(y, month, jd).strftime('%Y%m%d')
    return datetime(y, month, jd)

def date_to_JulianDate(dt, fmt):
    ################################################################
    #date_to_JulianDate('20180201', '%Y%m%d') -- 2018032
    #
    dt = datetime.strptime(dt, fmt)
    tt = dt.timetuple()
    return int('%d%03d' % (tt.tm_year, tt.tm_yday))


def fullname_to_datetime_for_DAAC3K(fullname):
    ################################################################
    #for modis hdf file, filename = 'DAAC_MOD04_3K/MOD04_3K.A2014003
    .0105.006.2015072123557.hdf'
    #
    import calendar
    fullname_el = fullname.split("/")
    filename_el = fullname_el[-1].split(".")
    y = int(filename_el[1][1:5])
    jd = int(filename_el[1][5:])
    month = 1
    while jd - calendar.monthrange(y, month)[1] > 0 and month <= 12:
        jd = jd - calendar.monthrange(y, month)[1]
        month += 1
    #print("filename_el: {}".format(filename_el))
    #print(y, month, jd, int(filename_el[2][:2]), int(filename_el[2][2:]))
    return datetime(y, month, jd, int(filename_el[2][:2]), int(filename_el
    [2][2:]))


def fullname_to_datetime_for_KOSC_MODIS_SST(fullname):
    ################################################################
    #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.
    hdf'
    #
```

```
80      from datetime import datetime
81
82      fullname_info = fullname.split('/')
83      fileinfo = fullname_info[-1].split('.')
84      filename_dt = datetime(int(fileinfo[-5]), int(fileinfo[-4][:2]), int(
        fileinfo[-4][2:]), int(fileinfo[-3][:2]), int(fileinfo[-3][2:]))
85      return filename_dt
86
87  def fullname_to_datetime_for_KOSC_AVHRR_SST_asc(fullname):
88      ###############################################################
89      #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.
        hdf'
90      #
91      from datetime import datetime
92
93      fullname_info = fullname.split('/')
94      fileinfo = fullname_info[-1].split('.')
95      filename_dt = datetime(int(fileinfo[0]), int(fileinfo[1][:2]), int(
        fileinfo[1][2:]), int(fileinfo[2][:2]), int(fileinfo[2][2:]))
96      return filename_dt
97
98  def fullname_to_datetime_for_L3_npyfile(fullname):
99      ###############################################################
100     #for modis hdf file, filename = '../folder/
        AVHRR_SST_20110901_20110902_115_145_20_55_0.5_alldata.npy'
101     #
102     from datetime import datetime
103
104     fullname_info = fullname.split('/')
105     fileinfo = fullname_info[-1].split('_')
106     filename_dt = datetime(int(fileinfo[-8][0:4]), int(fileinfo[-8][4:6]), int
        (fileinfo[-8][6:]))
107     return filename_dt
108
109 def fullname_to_datetime_for_KOSC_MODIS_hdf(fullname):
110     ###############################################################
111     #for modis hdf file, filename = '../folder/MYDOCT.2018.0724.0515.aqua-1.
        hdf'
112     #
113     from datetime import datetime
114
115     fullname_info = fullname.split('/')
116     fileinfo = fullname_info[-1].split('.')
117     filename_dt = datetime(int(fileinfo[1]), int(fileinfo[2][:2]), int(
        fileinfo[2][2:]), int(fileinfo[3][:2]), int(fileinfo[3][2:]))
118     return filename_dt
119
```

```
120
121 def draw_histogram_hdf(hdf_value, longitude, latitude, save_dir_name, fullname
        , DATAFIELD_NAME):
122     fullname_el = fullname.split("/")
123     import matplotlib.pyplot as plt
124     import numpy as np
125     plt.figure(figsize=(12, 8))
126     plt.title("Histogram of {0}: \n{1}\nmean : {2:.02f}, max: {3:.02f}, min:
        {4:.02f}\n\
127               longigude : {5:.02f}~{6:.02f}, latitude: {7:.02f}~{8:.02f}".
        format(DATAFIELD_NAME, fullname_el[-1],\
128                                       np.nanmean(hdf_value), np.nanmax(
        hdf_value), np.nanmin(hdf_value),\
129                                       np.nanmin(longitude), np.nanmax(
        longitude),\
130                                       np.nanmin(latitude), np.nanmax(latitude)
        ), fontsize=9)
131     plt.hist(hdf_value)
132     plt.grid(True)
133
134     return plt
135
136 def draw_histogram(hdf_value, longitude, latitude, save_dir_name, fullname,
        DATAFIELD_NAME):
137     fullname_el = fullname.split("/")
138     import matplotlib.pyplot as plt
139     import numpy as np
140     plt.figure(figsize=(12, 8))
141     plt.title("Histogram of {0}: \n{1}\nmean : {2:.02f}, max: {3:.02f}, min:
        {4:.02f}\n\
142               longigude : {5:.02f}~{6:.02f}, latitude: {7:.02f}~{8:.02f}".
        format(DATAFIELD_NAME, fullname_el[-1],\
143                                       np.nanmean(hdf_value), np.nanmax(
        hdf_value), np.nanmin(hdf_value),\
144                                       np.nanmin(longitude), np.nanmax(
        longitude),\
145                                       np.nanmin(latitude), np.nanmax(latitude)
        ), fontsize=9)
146     plt.hist(hdf_value)
147     plt.grid(True)
148     plt.savefig("{0}{1}_{2}_hist.png"\
149         .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
150     print("{0}{1}_{2}_hist.png is created..."\
151         .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
152     plt.close()
153     return None
154
```

```
155
156 def draw_map_MODIS_hdf(hdf_value, longitude, latitude, save_dir_name, fullname
        , DATAFIELD_NAME, Llon, Rlon, Slat, Nlat):
157     fullname_el = fullname.split("/")
158     import numpy as np
159     #if np.isnan(hdf_value).any() :
160     #    print("(np.isnan(hdf_value).any()) is true...")
161     #else :
162     from mpl_toolkits.basemap import Basemap
163     import matplotlib.pyplot as plt
164
165     plt.figure(figsize=(10, 10))
166
167     # sylender map
168     m = Basemap(projection='cyl', resolution='l', \
169                 llcrnrlat = Slat, urcrnrlat = Nlat, \
170                 llcrnrlon = Llon, urcrnrlon = Rlon)
171
172     m.drawcoastlines(linewidth=0.25, color='white')
173     m.drawcountries(linewidth=0.25, color='white')
174     m.fillcontinents(color='black', lake_color='black')
175     m.drawmapboundary()
176
177     m.drawparallels(np.arange(-90., 90., 10.), labels=[1, 0, 0, 0], color='
        white')
178     m.drawmeridians(np.arange(-180., 181., 15.), labels=[0, 0, 0, 1], color='
        white')
179
180     x, y = m(longitude, latitude) # convert to projection map
181
182     m.pcolormesh(x, y, hdf_value, vmin=0, vmax=40, cmap='coolwarm')
183     m.colorbar(fraction=0.0455, pad=0.044, ticks=(np.arange(-5, 40.1, step=5))
        )
184
185     plt.title('MODIS {}'.format(DATAFIELD_NAME), fontsize=20)
186
187     x1, y1 = m(Llon, Slat-1.5)
188     plt.text(x1, y1, "Maximun value: {0:.1f}\nMean value: {1:.1f}\nMin value:
        {2:.1f}\n"\
189             .format(np.nanmax(hdf_value), np.nanmean(hdf_value),
190                     np.nanmin(hdf_value)),
191             horizontalalignment='left',
192             verticalalignment='top',
193             fontsize=9, style='italic', wrap=True)
194
195     x2, y2 = m(Rlon, Slat-1.5)
```

```
196     plt.text(x2, y2, "created by guitar79@gs.hs.kr\nAVHRR SST procuct using
        KOSC data\n{}"\
197              .format(fullname_el[-1]),
198             horizontalalignment='right',
199             verticalalignment='top',
200             fontsize=10, style='italic', wrap=True)
201
202     return plt
203
204 def draw_map_SST_nc(hdf_value, longitude, latitude, save_dir_name, fullname,
        DATAFIELD_NAME, Llon, Rlon, Slat, Nlat):
205     fullname_el = fullname.split("/")
206     import numpy as np
207     #if np.isnan(hdf_value).any() :
208     #    print("(np.isnan(hdf_value).any()) is true...")
209     #else :
210     from mpl_toolkits.basemap import Basemap
211     import matplotlib.pyplot as plt
212
213     plt.figure(figsize=(10, 10))
214
215     # sylender map
216     m = Basemap(projection='cyl', resolution='l', \
217                 llcrnrlat = Slat, urcrnrlat = Nlat, \
218                 llcrnrlon = Llon, urcrnrlon = Rlon)
219
220     m.drawcoastlines(linewidth=0.25, color='white')
221     m.drawcountries(linewidth=0.25, color='white')
222     m.fillcontinents(color='black', lake_color='black')
223     m.drawmapboundary()
224
225     m.drawparallels(np.arange(-90., 90., 10.), labels=[1, 0, 0, 0], color='
        white')
226     m.drawmeridians(np.arange(-180., 181., 15.), labels=[0, 0, 0, 1], color='
        white')
227
228     lons,lats= np.meshgrid(longitude, latitude) # for this dataset, longitude
        is 0 through 360, so you need to subtract 180 to properly display on map
229     x,y = m(lons,lats)
230
231     #x, y = m(longitude, latitude) # convert to projection map
232
233     m.pcolormesh(x, y, hdf_value[0,:,:], vmin=0, vmax=40, cmap='coolwarm')
234     m.colorbar(fraction=0.0455, pad=0.044, ticks=(np.arange(-5, 40.1, step=5))
        )
235
236     plt.title('MODIS {}'.format(DATAFIELD_NAME), fontsize=20, y=1.03)
```

```
237
238     x1, y1 = m(Llon, Slat-1.5)
239     plt.text(x1, y1, "Maximun value: {0:.1f}\nMean value: {1:.1f}\nMin value:
        {2:.1f}\n"\
240             .format(np.nanmax(hdf_value), np.nanmean(hdf_value),
241                     np.nanmin(hdf_value)),
242             horizontalalignment='left',
243             verticalalignment='top',
244             fontsize=9, style='italic', wrap=True)
245
246     x2, y2 = m(Rlon, Slat-1.5)
247     plt.text(x2, y2, "created by guitar79@gs.hs.kr\nAVHRR SST procuct using
        KOSC data\n{}"\
248              .format(fullname_el[-1]),
249             horizontalalignment='right',
250             verticalalignment='top',
251             fontsize=10, style='italic', wrap=True)
252
253     return plt
254
255 def draw_histogram_SST_NC(SST, longitude, latitude, fullname, DATAFIELD_NAME):
256     fullname_el = fullname.split("/")
257     import matplotlib.pyplot as plt
258     import numpy as np
259     plt.figure(figsize=(12, 8))
260     #plt.title("Histogram of {0}: \n{1}\nmean : {2:.02f}, max: {3:.02f}, min:
        {4:.02f}\n\
261     #          longigude : {5:.02f}~{6:.02f}, latitude: {7:.02f}~{8:.02f}".
        format(DATAFIELD_NAME, fullname,\
262     #                                    np.nanmean(SST[0,:,:]), np.nanmax(SST
        [0,:,:]), np.nanmin(SST[0,:,:]),\
263     #                                    np.nanmin(longitude), np.nanmax(
        longitude),\
264     #                                    np.nanmin(latitude), np.nanmax(latitude
        )), fontsize=9)
265
266     plt.title("Histogram of {0}".format(DATAFIELD_NAME), fontsize=20, y=1.03)
267
268     ys, xs, patches =plt.hist(SST[0,:,:])
269     plt.xlim(int(np.min(xs)/10)*10, (int(np.max(xs)/10)+1)*10)
270     plt.ylim(int(np.min(ys)/10)*10, (int(np.max(ys)/10)+1)*10)
271
272     plt.grid(True)
273
274     plt.text(int(np.min(xs)/10)*10, (-1/np.max(ys))*25, "Maximun value: {0:.1f
        }\nMean value: {1:.1f}\nMin value: {2:.1f}\n"\
275             .format(np.nanmax(SST[0,:,:]), np.nanmean(SST[0,:,:]),
```

```
276                     np.nanmin(SST[0,:,:])),
277             horizontalalignment='left',
278             verticalalignment='top',
279             fontsize=9, style='italic', wrap=True)
280     plt.text((int(np.max(xs)/10)+1)*10, (-1/np.max(ys))*25, "created by
        guitar79@gs.hs.kr\nAVHRR SST procuct using KOSC data\n{}"\
281              .format(fullname_el[-1]),
282             horizontalalignment='right',
283             verticalalignment='top',
284             fontsize=10, style='italic', wrap=True)
285     #plt.text(min(xs), -0.25, "Maximun value: {0:.1f}\nMean value: {1:.1f}\
        nMin value: {2:.1f}\n"\
286     #         .format(np.nanmax(SST[0,:,:]), np.nanmean(SST[0,:,:]),
287     #                 np.nanmin(SST[0,:,:])),
288     #         horizontalalignment='left',
289     #         verticalalignment='top',
290     #         fontsize=9, style='italic', wrap=True)
291
292     #plt.text(np.nanmax(SST[0,:,:]), -0.25, "created by guitar79@gs.hs.kr\
        nAVHRR SST procuct using KOSC data\n{}"\
293     #          .format(fullname_el[-1]),
294     #         horizontalalignment='right',
295     #         verticalalignment='top',
296     #         fontsize=10, style='italic', wrap=True)
297     #plt.text(0, -0.25, "Maximun value: {0:.1f}\nMean value: {1:.1f}\nMin
        value: {2:.1f}\n"\
298     #         .format(np.nanmax(SST[0,:,:]), np.nanmean(SST[0,:,:]),
299     #                 np.nanmin(SST[0,:,:])),
300     #         horizontalalignment='left',
301     #         verticalalignment='top',
302     #         fontsize=9, style='italic', wrap=True)
303
304     return plt
305
306 def draw_map_AVHRR_SST_asc(df_AVHRR_sst, save_dir_name, fullname,
        DATAFIELD_NAME, Llon, Rlon, Slat, Nlat):
307     fullname_el = fullname.split("/")
308     import numpy as np
309     from mpl_toolkits.basemap import Basemap
310     import matplotlib.pyplot as plt
311
312     width = []
313     for i in range(len(df_AVHRR_sst)-1):
314         #print("index : {}".format(df_AVHRR_sst['latitude'].iloc[i]))
315         if abs(df_AVHRR_sst['latitude'].iloc[i] - df_AVHRR_sst['latitude'].
        iloc[i+1]) > 0.001 :
316             width.append(i)
```

```
            #print("index : {}".format(i))
        if i == 10000 : break
    longitude = df_AVHRR_sst['longitude'].to_numpy()
    longitude =  np.array(longitude, dtype=np.float32)
    longitude = longitude.reshape((longitude.shape[0]//(width[0]+1)), width
    [0]+1)
    latitude = df_AVHRR_sst['latitude'].to_numpy()
    latitude = np.array(latitude, dtype=np.float32)
    latitude = latitude.reshape((latitude.shape[0]//(width[0]+1)), width[0]+1)
    print("latitude.shape: {}".format(latitude.shape))
    print("type(latitude) : {}".format(type(latitude)))
    print("latitude: {}".format(latitude))
    print("np.nanmax(latitude): {}".format(np.nanmax(latitude)))
    print("np.nanmin(latitude): {}".format(np.nanmin(latitude)))

    sst = df_AVHRR_sst['sst'].to_numpy()
    sst = np.array(sst, dtype=np.float32)
    sst = sst.reshape((sst.shape[0]//(width[0]+1)), width[0]+1)

    #Plot data on the map
    print("="*80)
    print("Plotting data on the map")

    plt.figure(figsize=(10, 10))

    # sylender map
    m = Basemap(projection='cyl', resolution='l', \
                llcrnrlat = Slat, urcrnrlat = Nlat, \
                llcrnrlon = Llon, urcrnrlon = Rlon)

    m.drawcoastlines(linewidth=0.25, color='white')
    m.drawcountries(linewidth=0.25, color='white')
    m.fillcontinents(color='black', lake_color='black')
    m.drawmapboundary()

    m.drawparallels(np.arange(-90., 90., 10.), labels=[1, 0, 0, 0], color='
    white')
    m.drawmeridians(np.arange(-180., 181., 15.), labels=[0, 0, 0, 1], color='
    white')

    x, y = m(longitude, latitude) # convert to projection map

    m.pcolormesh(x, y, sst, vmin=0, vmax=40, cmap='coolwarm')
    m.colorbar(fraction=0.0455, pad=0.044, ticks=(np.arange(-5, 40.1, step=5))
    )

    plt.title('{}'.format(DATAFIELD_NAME), fontsize=20)
```

```
360
361     x1, y1 = m(Llon, Slat-1.5)
362     plt.text(x1, y1, "Maximun value: {0:.1f}\nMean value: {1:.1f}\nMin value:
        {2:.1f}\n"\
363             .format(np.nanmax(df_AVHRR_sst["sst"]), np.nanmean(df_AVHRR_sst["
        sst"]),
364                     np.nanmin(df_AVHRR_sst["sst"])),
365             horizontalalignment='left',
366             verticalalignment='top',
367             fontsize=9, style='italic', wrap=True)
368
369     x2, y2 = m(Rlon, Slat-1.5)
370     plt.text(x2, y2, "created by guitar79@gs.hs.kr\nAVHRR SST procuct using
        KOSC data\n{}"\
371              .format(fullname_el[-1]),
372             horizontalalignment='right',
373             verticalalignment='top',
374             fontsize=10, style='italic', wrap=True)
375
376     return plt
377
378
379 def draw_histogram_AVHRR_SST_asc(df_AVHRR_sst, save_dir_name, fullname,
        DATAFIELD_NAME):
380     fullname_el = fullname.split("/")
381     import matplotlib.pyplot as plt
382     import numpy as np
383     plt.figure(figsize=(12, 8))
384     plt.title("Histogram of {0}".format(DATAFIELD_NAME), fontsize=20)
385
386     #plt.title("Histogram of {0}: \n{1}\nmean : {2:.02f}, max: {3:.02f}, min:
        {4:.02f}\n\
387     #          longigude : {5:.02f}~{6:.02f}, latitude: {7:.02f}~{8:.02f}".
        format(DATAFIELD_NAME, fullname,\
388     #                                     np.nanmean(df_AVHRR_sst["sst"]), np.
        nanmax(df_AVHRR_sst["sst"]), np.nanmin(df_AVHRR_sst["sst"]),\
389     #                                     np.nanmin(df_AVHRR_sst["longitude"]),
        np.nanmax(df_AVHRR_sst["longitude"]),\
390     #                                     np.nanmin(df_AVHRR_sst["latitude"]), np
        .nanmax(df_AVHRR_sst["latitude"])), fontsize=9)
391     plt.hist(df_AVHRR_sst["sst"])
392     plt.grid(True)
393
394     plt.text(0, -0.2, "Maximun value: {0:.1f}\nMean value: {1:.1f}\nMin value:
         {2:.1f}\n"\
395             .format(np.nanmax(df_AVHRR_sst["sst"]), np.nanmean(df_AVHRR_sst["
        sst"]),
```

```
396                     np.nanmin(df_AVHRR_sst["sst"])),
397             horizontalalignment='left',
398             verticalalignment='top',
399             fontsize=9, style='italic', wrap=True)
400
401     plt.text(np.nanmax(df_AVHRR_sst["sst"]), -0.2, "created by guitar79@gs.hs.
        kr\nAVHRR SST procuct using KOSC data\n{}"\
402              .format(fullname_el[-1]),
403             horizontalalignment='right',
404             verticalalignment='top',
405             fontsize=10, style='italic', wrap=True)
406
407
408
409     return plt
410
411 def draw_histogram_AVHRR_SST_asc1(df_AVHRR_sst, save_dir_name, fullname,
        DATAFIELD_NAME):
412     fullname_el = fullname.split("/")
413     import matplotlib.pyplot as plt
414     import numpy as np
415     plt.figure(figsize=(12, 8))
416     plt.title("Histogram of {0}: \n{1}\nmean : {2:.02f}, max: {3:.02f}, min:
        {4:.02f}\n\
417               longigude : {5:.02f}~{6:.02f}, latitude: {7:.02f}~{8:.02f}".
        format(DATAFIELD_NAME, fullname,\
418                                       np.nanmean(df_AVHRR_sst["sst"]), np.
        nanmax(df_AVHRR_sst["sst"]), np.nanmin(df_AVHRR_sst["sst"]),\
419                                       np.nanmin(df_AVHRR_sst["longitude"]), np
        .nanmax(df_AVHRR_sst["longitude"]),\
420                                       np.nanmin(df_AVHRR_sst["latitude"]), np.
        nanmax(df_AVHRR_sst["latitude"])), fontsize=9)
421     plt.hist(df_AVHRR_sst["sst"])
422     plt.grid(True)
423
424     #plt.savefig("{}_{}_hist.png".format(fullname[:-4], DATAFIELD_NAME))
425     #print("{}_{}_hist.png is created...".format(fullname[:-4], DATAFIELD_NAME
        ))
426     plt.savefig("{0}{1}_{2}_hist.png"\
427         .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
428     print("{0}{1}_{2}_hist.png is created..."\
429         .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
430     plt.close()
431     return None
432
433 def npy_filename_to_fileinfo(fullname):
434     ##############################################################
```

```
    # for modis hdf file, filename = 'DAAC_MOD04_3K/daily/
    sst_20110901_20110902_110_150_10_60_0.05_alldata.npy'
    #
    fileinfo = fullname.split('_')
    start_date = fileinfo[-8]
    end_date = fileinfo[-7]
    Llon = fileinfo[-6]
    Rlon = fileinfo[-5]
    Slat = fileinfo[-4]
    Nlat = fileinfo[-3]
    resolution = fileinfo[-2]
    return start_date, end_date, Llon, Rlon, Slat, Nlat, resolution

def getFullnameListOfallFiles(dirName):
    ##############################################3
    import os
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = sorted(os.listdir(dirName))
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + getFullnameListOfallFiles(fullPath)
        else:
            allFiles.append(fullPath)
    return allFiles


def calculate_mean_using_result_array(result_array):
    mean_array = result_array.copy()
    cnt_array = result_array.copy()
    for i in range(np.shape(result_array)[0]):
        for j in range(np.shape(result_array)[1]):

            if len(result_array[i][j])>0: mean_array[i][j] = np.mean(
    result_array[i][j])
            else : mean_array[i][j] = np.nan
            cnt_array[i][j] = len(result_array[i][j])

    mean_array = np.array(mean_array)
    cnt_array = np.array(cnt_array)
    return mean_array, cnt_array

```

```
480 def make_grid_array(Llon, Rlon, Slat, Nlat, resolution) :
481     ###############################################################
482     # Llon, Rlon = 90, 150
483     # Slat, Nlat = 10, 60
484     # resolution = 0.025
485     #
486 
487     import numpy as np
488 
489     ni = np.int((Rlon-Llon)/resolution+1.00)
490     nj = np.int((Nlat-Slat)/resolution+1.00)
491     array_data = []
492     for i in range(ni):
493         line_data = []
494         for j in range(nj):
495             line_data.append([])
496         array_data.append(line_data)
497 
498     return array_data
499 
500 
501 def make_grid_array1(Llon, Rlon, Slat, Nlat, resolution) :
502     ###############################################################
503     # Llon, Rlon = 90, 150
504     # Slat, Nlat = 10, 60
505     # resolution = 0.025
506     #
507 
508     import numpy as np
509 
510     ni = np.int((Rlon-Llon)/resolution+1.00)
511     nj = np.int((Nlat-Slat)/resolution+1.00)
512     array_lon = []
513     array_lat = []
514     array_data = []
515     for i in range(ni):
516         line_lon = []
517         line_lat = []
518         line_data = []
519         for j in range(nj):
520             line_lon.append(Llon+resolution*i)
521             line_lat.append(Nlat-resolution*j)
522             line_data.append([])
523         array_lon.append(line_lon)
524         array_lat.append(line_lat)
525         array_data.append(line_data)
526     array_lon = np.array(array_lon)
```

```
    array_lat = np.array(array_lat)

    return array_lon, array_lat, array_data



def read_MODIS_hdf_to_ndarray(fullname, DATAFIELD_NAME):
    ############################################################
    #
    #
    import numpy as np
    from pyhdf.SD import SD, SDC
    hdf = SD(fullname, SDC.READ)

    # Read AOD dataset.
    if DATAFIELD_NAME.upper() in hdf.datasets() :
        DATAFIELD_NAME = DATAFIELD_NAME.upper()

    if DATAFIELD_NAME in hdf.datasets() :
        hdf_raw = hdf.select(DATAFIELD_NAME)
        print("found data set of {}: {}".format(DATAFIELD_NAME, hdf_raw))

    else :
        print("There is no data set of {}: {}".format(DATAFIELD_NAME, hdf_raw)
    )
        hdf_raw = np.arange(0)

    # Read geolocation dataset.
    if 'Latitude' in hdf.datasets() and 'Longitude' in hdf.datasets():
        lat = hdf.select('Latitude')
        latitude = lat[:,:]
        lon = hdf.select('Longitude')
        longitude = lon[:,:]

    elif 'Latitude'.lower() in hdf.datasets() and 'Longitude'.lower()  in hdf.
    datasets():
        lat = hdf.select('Latitude'.lower())
        latitude = lat[:,:]
        lon = hdf.select('Longitude'.lower())
        longitude = lon[:,:]
    else :
        latitude, longitude  \
            = np.arange(0), np.arange(0)

    if 'cntl_pt_cols' in hdf.datasets() and 'cntl_pt_rows' in hdf.datasets():
        cntl_pt_cols = hdf.select('cntl_pt_cols')
        cntl_pt_cols = cntl_pt_cols[:]
```

```
572         cntl_pt_rows = hdf.select('cntl_pt_rows')
573         cntl_pt_rows = cntl_pt_rows[:]
574     else :
575         cntl_pt_cols, cntl_pt_rows =  np.arange(0), np.arange(0)
576 
577     return hdf_raw, latitude, longitude, cntl_pt_cols, cntl_pt_rows
578 
579 
580 
581 def read_MODIS_hdf_and_make_statistics_array(dir_name, DATAFIELD_NAME,
        proc_date,
582                                 resolution, Llon, Rlon, Slat, Nlat):
583 
584     proc_start_date = proc_date[0]
585     proc_end_date = proc_date[1]
586     thread_number = proc_date[2]
587     processing_log = '#This file is created using python \n' \
588                 '#https://github.com/guitar79/KOSC_MODIS_SST_Python \n' \
589                 + '#start date = ' + str(proc_date[0]) +'\n'\
590                 + '#end date = ' + str(proc_date[1]) +'\n'
591 
592     #convert start_date and end_date to date type
593     start_date = datetime(int(proc_start_date[:4]),
594                           int(proc_start_date[4:6]),
595                           int(proc_start_date[6:8]))
596     end_date = datetime(int(proc_end_date[:4]),
597                         int(proc_end_date[4:6]),
598                         int(proc_end_date[6:8]))
599 
600     processing_log += '#Llon =' + str(Llon) + '\n' \
601     + '#Rlon =' + str(Rlon) + '\n' \
602     + '#Slat =' + str(Slat) + '\n' \
603     + '#Nlat =' + str(Nlat) + '\n' \
604     + '#resolution =' + str(resolution) + '\n'
605 
606     print("{0}-{1} Start making grid arrays...\n".format(proc_start_date,
        proc_end_date))
607     ni = np.int((Rlon-Llon)/resolution+1.00)
608     nj = np.int((Nlat-Slat)/resolution+1.00)
609     array_lon = []
610     array_lat = []
611     array_data = []
612     for i in range(ni):
613         line_lon = []
614         line_lat = []
615         line_data = []
616         for j in range(nj):
```

```
            line_lon.append(Llon+resolution*i)
            line_lat.append(Nlat-resolution*j)
            line_data.append([])
        array_lon.append(line_lon)
        array_lat.append(line_lat)
        array_data.append(line_data)
    array_lat = np.array(array_lat)
    array_lon = np.array(array_lon)
    print('Grid arrays are created..........\n')

    total_data_cnt = 0
    file_no = 0
    processing_log += '#processing file list\n'
    processing_log += '#No, data_count, filename \n'

    result_array = np.zeros((1, 1, 1))
    fullnames = sorted(glob(os.path.join(dir_name, '*.hdf')))
    if not fullnames  :
        for fullname in fullnames:
            result_array = array_data
            file_date = fullname_to_datetime_for_MODIS_3K(fullname)
            #print('fileinfo', file_date)

            if file_date >= start_date \
                and file_date < end_date :

                try:
                    print('reading file {0}\n'.format(fullname))
                    hdf = SD(fullname, SDC.READ)
                    # Read AOD dataset.
                    hdf_raw = hdf.select(DATAFIELD_NAME)
                    hdf_data = hdf_raw[:,:]
                    scale_factor = hdf_raw.attributes()['scale_factor']
                    offset = hdf_raw.attributes()['add_offset']
                    hdf_value = hdf_data * scale_factor + offset
                    hdf_value[hdf_value < 0] = np.nan
                    hdf_value = np.asarray(hdf_value)

                    # Read geolocation dataset.
                    lat = hdf.select('Latitude')
                    latitude = lat[:,:]
                    lon = hdf.select('Longitude')
                    longitude = lon[:,:]
                except Exception as err :
                    print("Something got wrecked : {}".format(err))
                    continue

```

```
664                 if np.shape(longitude) != np.shape(latitude) or np.shape(
    latitude) != np.shape(hdf_value) :
665                     print('data shape is different!! \n')
666                     print('='*80)
667                 else :
668                     lon_cood = np.array(((longitude-Llon)/resolution*100//100)
    , dtype=np.uint16)
669                     lat_cood = np.array(((Nlat-latitude)/resolution*100//100),
     dtype=np.uint16)
670                     data_cnt = 0
671                     for i in range(np.shape(lon_cood)[0]) :
672                         for j in range(np.shape(lon_cood)[1]) :
673                             if int(lon_cood[i][j]) < np.shape(array_lon)[0] \
674                                 and int(lat_cood[i][j]) < np.shape(array_lon)
    [1] \
675                                 and not np.isnan(hdf_value[i][j]) :
676                                 data_cnt += 1 #for debug
677                                 result_array[int(lon_cood[i][j])][int(lat_cood
    [i][j])].append(hdf_value[i][j])
678                     file_no += 1
679                     total_data_cnt += data_cnt
680                 processing_log += str(file_no) + ',' + str(data_cnt) +',' +
    str(fullname) + '\n'
681                 print(thread_number,  proc_date[0], 'number of files: ',
682                       file_no, 'tatal data cnt :' , data_cnt)
683         processing_log += '#total data number =' + str(total_data_cnt) + '\n'
684
685     else  :
686         print("No file exist...")
687
688     return result_array, processing_log
689
690
691
692
693 def read_MODIS_SST_hdf_and_array_by_date(save_dir_name, dir_name, proc_date,
694                                 resolution, Llon, Rlon, Slat, Nlat):
695     add_log = True
696     if add_log == True :
697         log_file = 'read_MODIS_AOD_hdf_and_array_by_date.log'
698         err_log_file = 'read_MODIS_AOD_hdf_and_array_by_date_err.log'
699
700     proc_start_date = proc_date[0]
701     proc_end_date = proc_date[1]
702     thread_number = proc_date[2]
703     processing_log = '#This file is created using python \n' \
704                 '#https://github.com/guitar79/MODIS_AOD \n' \
```

```
                + '#start date = ' + str(proc_date[0]) +'\n'\
                + '#end date = ' + str(proc_date[1]) +'\n'
    #variables for downloading
    start_date = datetime(int(proc_start_date[:4]),
                          int(proc_start_date[4:6]),
                          int(proc_start_date[6:8])) #convert startdate to
    date type
    end_date = datetime(int(proc_end_date[:4]),
                        int(proc_end_date[4:6]),
                        int(proc_end_date[6:8])) #convert startdate to date
    type

    print('checking... {0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy\n'\
          .format(save_dir_name, proc_start_date, proc_end_date,
          str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)))
    if os.path.exists('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy'\
          .format(save_dir_name, proc_start_date, proc_end_date,
          str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))) \
        and os.path.exists('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_info.txt'\
          .format(save_dir_name, proc_start_date, proc_end_date,
          str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))):

        print('='*80)
        write_log(log_file, '{8} ::: {0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}
    files are already exist'\
                  .format(save_dir_name, proc_start_date, proc_end_date,
                  str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution),
     datetime.now()))
        return 0

    else :
        processing_log += '#Llon =' + str(Llon) + '\n' \
        + '#Rlon =' + str(Rlon) + '\n' \
        + '#Slat =' + str(Slat) + '\n' \
        + '#Nlat =' + str(Nlat) + '\n' \
        + '#resolution =' + str(resolution) + '\n'

        print('{0}-{1} Start making grid arrays...\n'\
              .format(proc_start_date, proc_end_date))
        ni = np.int((Rlon-Llon)/resolution+1.00)
        nj = np.int((Nlat-Slat)/resolution+1.00)
        array_lon = []
        array_lat = []
        array_data = []
        for i in range(ni):
            line_lon = []
            line_lat = []
```

```
            line_data = []
            for j in range(nj):
                line_lon.append(Llon+resolution*i)
                line_lat.append(Nlat-resolution*j)
                line_data.append([])
            array_lon.append(line_lon)
            array_lat.append(line_lat)
            array_data.append(line_data)
        array_lat = np.array(array_lat)
        array_lon = np.array(array_lon)
        print('grid arrays are created..........\n')

        total_data_cnt = 0
        file_no=0
        processing_log += '#processing file list\n'
        processing_log += '#No, data_count, filename \n'

        result_array = np.zeros((1, 1, 1))
        for fullname in sorted(glob(os.path.join(dir_name, '*.hdf'))):

            result_array = array_data
            file_date = fullname_to_datetime_for_MODIS_3K(fullname)
            #print('fileinfo', file_date)

            if file_date >= start_date \
                and file_date < end_date :

                try:
                    print('reading file {0}\n'.format(fullname))
                    hdf = SD(fullname, SDC.READ)
                    # Read AOD dataset.
                    DATAFIELD_NAME = 'Optical_Depth_Land_And_Ocean'
                    hdf_raw = hdf.select(DATAFIELD_NAME)
                    hdf_data = hdf_raw[:,:]
                    scale_factor = hdf_raw.attributes()['scale_factor']
                    offset = hdf_raw.attributes()['add_offset']
                    hdf_value = hdf_data * scale_factor + offset
                    hdf_value[hdf_value < 0] = np.nan
                    hdf_value = np.asarray(hdf_value)

                    # Read geolocation dataset.
                    lat = hdf.select('Latitude')
                    latitude = lat[:,:]
                    lon = hdf.select('Longitude')
                    longitude = lon[:,:]
                except Exception as err :
                    print("Something got wrecked \n")
```

```
                    write_log(err_log_file, '{2} ::: {0} with {1}'\
                      .format(err, fullname, datetime.now()))
                    continue

            if np.shape(longitude) != np.shape(latitude) or np.shape(
latitude) != np.shape(hdf_value) :
                print('data shape is different!! \n')
                print('='*80)
            else :
                lon_cood = np.array(((longitude-Llon)/resolution*100//100)
, dtype=np.uint16)
                lat_cood = np.array(((Nlat-latitude)/resolution*100//100),
 dtype=np.uint16)
                data_cnt = 0
                for i in range(np.shape(lon_cood)[0]) :
                    for j in range(np.shape(lon_cood)[1]) :
                        if int(lon_cood[i][j]) < np.shape(array_lon)[0] \
                            and int(lat_cood[i][j]) < np.shape(array_lon)
[1] \
                            and not np.isnan(hdf_value[i][j]) :
                            data_cnt += 1 #for debug
                            result_array[int(lon_cood[i][j])][int(lat_cood
[i][j])].append(hdf_value[i][j])
                file_no += 1
                total_data_cnt += data_cnt
            processing_log += str(file_no) + ',' + str(data_cnt) +',' +
str(fullname) + '\n'
            print(thread_number,  proc_date[0], 'number of files: ',
                  file_no, 'tatal data cnt :' , data_cnt)
    processing_log += '#total data number =' + str(total_data_cnt) + '\n'

    np.save('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_result.npy'\
            .format(save_dir_name, proc_start_date, proc_end_date,
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)),
result_array)

    with open('{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7}_info.txt'\
              .format(save_dir_name, proc_start_date, proc_end_date,
              str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))
, 'w') as f:
        f.write(processing_log)
    print('#'*60)
    write_log(log_file, '{0}AOD_3K_{1}_{2}_{3}_{4}_{5}_{6}_{7} files are
is created.'\
          .format(save_dir_name, proc_start_date, proc_end_date,
              str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))
)
```

```
    return 0 # Return a dummy value
    # Putting large values in Queue was slow than expected(~10min)
    #return result_array, processing_log
```

V.2 1.daily_classify_using_AVHRR_asc_SST.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''
#####################################################################
#runfile('./classify_AVHRR_asc_SST-01.py', 'daily 0.1 2019', wdir='./MODIS_hdf_Python/')
#cd '/mnt/14TB1/RS-data/KOSC/MODIS_hdf_Python' && for yr in {2011..2020}; do python classify_AVHRR_asc_SST-01.py daily 0.05 $yr; done
#conda activate MODIS_hdf_Python_env && cd '/mnt/14TB1/RS-data/KOSC/MODIS_hdf_Python' && python classify_AVHRR_asc_SST.py daily 0.01 2011
#conda activate MODIS_hdf_Python_env && cd /mnt/Rdata/RS-data/KOSC/MODIS_hdf_Python/ && python classify_AVHRR_asc_SST.py daily 1.0 2019
'''



from glob import glob
from datetime import datetime
import numpy as np
import os
import sys
import MODIS_hdf_utilities

arg_mode = True
arg_mode =  False

log_file = os.path.basename(__file__)[:-3]+".log"
err_log_file = os.path.basename(__file__)[:-3]+"_err.log"
print ("log_file: {}".format(log_file))
print ("err_log_file: {}".format(err_log_file))

if arg_mode == True :
    from sys import argv # input option
    print("argv: {}".format(argv))

    if len(argv) < 3 :
        print ("len(argv) < 2\nPlease input L3_perid and year \n ex) aaa.py 0.1 2016")
        sys.exit()
    elif len(argv) > 3 :
```

```
        print ("len(argv) > 2\nPlease input L3_perid and year \n ex) aaa.py 0.1 2016")
        sys.exit()
    else :
        L3_perid, resolution, year = 'daily', argv[1], float(argv[2])
        print("{}, {}, processing started...".format(argv[1], argv[2]))
        sys.exit()
else :

    L3_perid, resolution, year = 'daily', 0.5, 2019

# Set Datafield name
DATAFIELD_NAME = "AVHRR_SST"

#Set lon, lat, resolution
Llon, Rlon = 115, 145
Slat, Nlat = 20, 55
#L3_perid, resolution, yr = "daily", 0.1, 2019

#set directory
base_dir_name = '../L2_AVHRR_SST/'
save_dir_name = "../L3_{0}/{0}_{1}_{2}_{3}_{4}_{5}_date/".format(DATAFIELD_NAME, str(Llon), str(Rlon),
                                                                 str(Slat), str(Nlat), str(resolution))
if not os.path.exists(save_dir_name):
    os.makedirs(save_dir_name)
    print ('*'*80)
    print (save_dir_name, 'is created')
else :
    print ('*'*80)
    print (save_dir_name, 'is exist')

proc_dates = []

#make processing period tuple
from dateutil.relativedelta import relativedelta
s_start_date = datetime(year, 1, 1) #convert startdate to date type
s_end_date = datetime(year+1, 1, 1)

k=0
date1 = s_start_date
date2 = s_start_date

while date2 < s_end_date :
    k += 1

```

```
    date2 = date1 + relativedelta(days=1)

    date = (date1, date2, k)
    proc_dates.append(date)
    date1 = date2

#### make dataframe from file list
fullnames = sorted(glob(os.path.join(base_dir_name, '*.asc')))

fullnames_dt = []
for fullname in fullnames :
    fullnames_dt.append(MODIS_hdf_utilities.
    fullname_to_datetime_for_KOSC_AVHRR_SST_asc(fullname))

import pandas as pd

len(fullnames)
len(fullnames_dt)

# Calling DataFrame constructor on list
df = pd.DataFrame({'fullname':fullnames,'fullname_dt':fullnames_dt})
df.index = df['fullname_dt']
print("df:\n{}".format(df))

#proc_date = proc_dates[0]
for proc_date in proc_dates[:]:
    #proc_date = proc_dates[0]
    df_proc = df[(df['fullname_dt'] >= proc_date[0]) & (df['fullname_dt'] <
    proc_date[1])]

    #check file exist??
    if os.path.exists('{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8}_alldata.npy'\
            .format(save_dir_name, DATAFIELD_NAME, proc_date[0].strftime('%Y%m
    %d'), proc_date[1].strftime('%Y%m%d'),
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)))\
        and os.path.exists('{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8}_info.txt'\
            .format(save_dir_name, DATAFIELD_NAME, proc_date[0].strftime('%Y%m
    %d'), proc_date[1].strftime('%Y%m%d'),
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))) :

        print(('{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8} files are exist...'
            .format(save_dir_name, DATAFIELD_NAME, proc_date[0].strftime('%Y%m
    %d'), proc_date[1].strftime('%Y%m%d'),
            str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution))))

    else :

```

```
121         if len(df_proc) == 0 :
122             print("There is no data in {0} - {1} ...\n"\
123                   .format(proc_date[0].strftime('%Y%m%d'), proc_date[1].
    strftime('%Y%m%d')))
124 
125         else :
126 
127             print("df_proc: {}".format(df_proc))
128 
129             processing_log = "#This file is created using Python : https://
    github.com/guitar79/MODIS_hdf_Python\n"
130             processing_log += "#L3_perid = {}, start date = {}, end date = {}\
    n"\
131                 .format(L3_perid, proc_date[0].strftime('%Y%m%d'), proc_date
    [1].strftime('%Y%m%d'))
132 
133             processing_log += "#Llon = {}, Rlon = {}, Slat = {}, Nlat = {},
    resolution = {}\n"\
134                 .format(str(Llon), str(Rlon), str(Slat), str(Nlat), str(
    resolution))
135 
136             # make array_data
137             print("{0}-{1} Start making grid arrays...\n".\
138                   format(proc_date[0].strftime('%Y%m%d'), proc_date[1].
    strftime('%Y%m%d')))
139             array_data = MODIS_hdf_utilities.make_grid_array(Llon, Rlon, Slat,
     Nlat, resolution)
140             print('Grid arrays are created..........\n')
141 
142             total_data_cnt = 0
143             file_no = 0
144             processing_log += "#processing file Num : {}\n".format(len(df_proc
    ["fullname"]))
145             processing_log += "#processing file list\n"
146             processing_log += "#file No, total_data_dount, data_count,
    filename, mean(sst), max(sst), min(sst), min(longitude), max(longitude),
    min(latitude), max(latitude)\n"
147             array_alldata = array_data.copy()
148             print('array_alldata is copied..........\n')
149 
150             for fullname in df_proc["fullname"] :
151 
152                 file_no += 1
153 
154                 try :
155 
156                     #fullname = df_proc["fullname"][0]
```

```
157                     fullname_el = fullname.split("/")
158                     print("Reading ascii file {0}\n".format(fullname))
159                     df_AVHRR_sst = pd.read_table("{}".format(fullname), sep='\
    t', header=None, index_col=0,
160                                          names = ['index', 'latitude', '
    longitude', 'sst'],
161                                          engine='python')
162                     df_AVHRR_sst = df_AVHRR_sst.drop(df_AVHRR_sst[df_AVHRR_sst
    .sst == "***"].index)
163                     #df_AVHRR_sst.loc[df_AVHRR_sst.sst == "***", ['sst']] = np
    .nan
164                     df_AVHRR_sst["sst"] = df_AVHRR_sst.sst.astype("float64")
165                     df_AVHRR_sst["longitude"] = df_AVHRR_sst.longitude.astype(
    "float64")
166                     df_AVHRR_sst["latitude"] = df_AVHRR_sst.latitude.astype("
    float64")
167                     print("df_AVHRR_sst : {}".format(df_AVHRR_sst))
168
169                     #check dimension
170                     if len(df_AVHRR_sst) == 0 :
171                         processing_log += "{0}, 0, 0, {1}, \n"\
172                             .format(str(file_no), str(fullname))
173                         print("There is no sst data...")
174
175                     else :
176                         df_AVHRR_sst = df_AVHRR_sst.drop(df_AVHRR_sst[
    df_AVHRR_sst.longitude < Llon].index)
177                         df_AVHRR_sst = df_AVHRR_sst.drop(df_AVHRR_sst[
    df_AVHRR_sst.longitude > Rlon].index)
178                         df_AVHRR_sst = df_AVHRR_sst.drop(df_AVHRR_sst[
    df_AVHRR_sst.latitude > Nlat].index)
179                         df_AVHRR_sst = df_AVHRR_sst.drop(df_AVHRR_sst[
    df_AVHRR_sst.latitude < Slat].index)
180                         df_AVHRR_sst["lon_cood"] = (((df_AVHRR_sst["longitude"
    ]-Llon)/resolution*100)//100)
181                         df_AVHRR_sst["lat_cood"] = (((Nlat-df_AVHRR_sst["
    latitude"])/resolution*100)//100)
182                         df_AVHRR_sst["lon_cood"] = df_AVHRR_sst.lon_cood.
    astype("int16")
183                         df_AVHRR_sst["lat_cood"] = df_AVHRR_sst.lat_cood.
    astype("int16")
184                         df_AVHRR_sst = df_AVHRR_sst.dropna()
185
186                         data_cnt = 0
187                         NaN_cnt = 0
188
189                         for index, row in df_AVHRR_sst.iterrows():
```

```
                            data_cnt += 1
                            #array_alldata[int(lon_cood[i][j])][int(lat_cood[i][j])].append(hdf_value[i][j])
                            array_alldata[df_AVHRR_sst.lon_cood[index]][df_AVHRR_sst.lat_cood[index]].append((fullname_el[-1], df_AVHRR_sst.sst[index]))
                            print("array_alldata[{}][{}].append({}, {})"\
                                  .format(df_AVHRR_sst.lon_cood[index], df_AVHRR_sst.lat_cood[index], fullname_el[-1], df_AVHRR_sst.sst[index]))

                            #array_alldata[df_AVHRR_sst.lon_cood[index]][df_AVHRR_sst.lat_cood[index]].append(df_AVHRR_sst.sst[index])
                            #print("array_alldata[{}][{}].append({})"\
                            #      .format(df_AVHRR_sst.lon_cood[index], df_AVHRR_sst.lat_cood[index], df_AVHRR_sst.sst[index]))

                            print("{} data added...".format(data_cnt))

                        total_data_cnt += data_cnt

                        processing_log += "{0}, {1}, {2}, {3}, {4:.02f}, {5:.02f}, {6:.02f}, {7:.02f}, {8:.02f}, {9:.02f}, {10:.02f}\n"\
                            .format(str(file_no), str(total_data_cnt), str(data_cnt), str(fullname),
                                    np.nanmean(df_AVHRR_sst["sst"]), np.nanmax(df_AVHRR_sst["sst"]), np.nanmin(df_AVHRR_sst["sst"]),
                                    np.nanmin(df_AVHRR_sst["longitude"]), np.nanmax(df_AVHRR_sst["longitude"]),
                                    np.nanmin(df_AVHRR_sst["latitude"]), np.nanmax(df_AVHRR_sst["latitude"]))

                except Exception as err :
                    MODIS_hdf_utilities.write_log(err_log_file, err)
                    continue

            processing_log += "#processing finished!!!\n"
            # print("array_alldata: {}".format(array_alldata))
            print("prodessing_log: {}".format(processing_log))

            array_alldata = np.array(array_alldata)
            #array_alldata1 = np.array(array_alldata)
            #array_alldata[:,:,0] = [1]
            #array_alldata = np.nan
            #array_alldata[array_alldata==np.empty]=np.nan

            print("array_alldata: \n{}".format(array_alldata))
            print("array_alldata.shape: {}".format(array_alldata.shape))
```

```
            np.save('{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8}_alldata.npy' \
                    .format(save_dir_name, DATAFIELD_NAME,
                    proc_date[0].strftime('%Y%m%d'), proc_date[1].strftime('%Y
    %m%d'),
                    str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution
    )), array_alldata)

            with open('{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8}_info.txt' \
                    .format(save_dir_name, DATAFIELD_NAME,
                    proc_date[0].strftime('%Y%m%d'), proc_date[1].strftime('%Y
    %m%d'),
                    str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution
    )), 'w') as f:
                f.write(processing_log)

            print('#' * 60)
            MODIS_hdf_utilities.write_log(log_file,
                '{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8} files are is created.' \
                .format(save_dir_name, DATAFIELD_NAME,
                proc_date[0].strftime('%Y%m%d'), proc_date[1].strftime('%Y%m%d
    '),
                str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution)))
```

V.3 2.statistics_AVHRR_asc_SST_alldata_and_creating_NCfile.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''
#####################################################################
#runfile('./classify_AVHRR_asc_SST-01.py', 'daily 0.1 2019', wdir='./
    MODIS_hdf_Python/')
#cd '/mnt/14TB1/RS-data/KOSC/MODIS_hdf_Python' && for yr in {2011..2020}; do
    python classify_AVHRR_asc_SST-01.py daily 0.05 $yr; done
#conda activate MODIS_hdf_Python_env && cd '/mnt/14TB1/RS-data/KOSC/
    MODIS_hdf_Python' && python classify_AVHRR_asc_SST.py daily 0.01 2011
#conda activate MODIS_hdf_Python_env && cd /mnt/Rdata/RS-data/KOSC/
    MODIS_hdf_Python/ && python 2.
    statistics_AVHRR_asc_SST_alldata_and_creating_NCfile.py daily
'''

from glob import glob
from datetime import datetime
import numpy as np
import netCDF4 as nc
import os
import sys
```

```
17 import MODIS_hdf_utilities
18
19 log_file = os.path.basename(__file__)[:-3]+".log"
20 err_log_file = os.path.basename(__file__)[:-3]+"_err.log"
21 print ("log_file: {}".format(log_file))
22 print ("err_log_file: {}".format(err_log_file))
23
24 arg_mode = True
25 arg_mode = False
26
27 if arg_mode == True :
28     from sys import argv # input option
29     print("argv: {}".format(argv))
30
31     if len(argv) < 2 :
32         print ("len(argv) < 2\nPlease input L3_perid and year \n ex) aaa.py
      daily")
33         sys.exit()
34     elif len(argv) > 2 :
35         print ("len(argv) > 2\nPlease input L3_perid and year \n ex) aaa.py
      daily")
36         sys.exit()
37     elif argv[1] == 'daily' or argv[1] == 'weekly' or argv[1] == 'monthly' :
38         L3_perid = argv[1]
39         print("{} processing started...".format(argv[1]))
40     else :
41         print("Please input L3_perid \n ex) aaa.py daily")
42         sys.exit()
43 else :
44     L3_perid, resolution = 'weekly', 0.5
45
46
47 # Set Datafield name
48 DATAFIELD_NAME = "AVHRR_SST"
49
50 #Set lon, lat, resolution
51 Llon, Rlon = 115, 145
52 Slat, Nlat = 20, 55
53
54 #set directory
55 base_dir_name = "../L3_{0}/{0}_{1}_{2}_{3}_{4}_{5}_date/".format(
      DATAFIELD_NAME, str(Llon), str(Rlon),
56                                                     str(Slat), str(Nlat),
      str(resolution))
57 save_dir_name = "../L3_{0}/{0}_{1}_{2}_{3}_{4}_{5}_{6}/".format(DATAFIELD_NAME
      , str(Llon), str(Rlon),
```

```
                                                    str(Slat), str(Nlat),
    str(resolution), L3_perid)

if not os.path.exists(save_dir_name):
    os.makedirs(save_dir_name)
    print('*' * 80)
    print(save_dir_name, 'is created')
else:
    print('*' * 80)
    print(save_dir_name, 'is exist')

proc_dates = []

# make processing period tuple
from dateutil.relativedelta import relativedelta
s_start_date = datetime(2000, 1, 1)  # convert startdate to date type
s_end_date = datetime(2022, 1, 1)

k = 0
date1 = s_start_date
date2 = s_start_date

while date2 < s_end_date:
    k += 1
    if L3_perid == 'daily':
        date2 = date1 + relativedelta(days=1)
    elif L3_perid == 'weekly':
        date2 = date1 + relativedelta(days=8)
    elif L3_perid == 'monthly':
        date2 = date1 + relativedelta(months=1)

    date = (date1, date2, k)
    proc_dates.append(date)
    date1 = date2

#### make dataframe from file list
fullnames = sorted(glob(os.path.join(base_dir_name, '*alldata.npy')))
print("len(fullnames): {}".format(len(fullnames)))

fullnames_dt = []
for fullname in fullnames :
    fullnames_dt.append(MODIS_hdf_utilities.
    fullname_to_datetime_for_L3_npyfile(fullname))

import pandas as pd

# Calling DataFrame constructor on list
```

```
103 df = pd.DataFrame({'fullname': fullnames, 'fullname_dt': fullnames_dt})
104 df.index = df['fullname_dt']
105 print("fullnames_dt:\n{}".format(fullnames_dt))
106 print("len(fullnames_dt):\n{}".format(len(fullnames_dt)))
107
108 for proc_date in proc_dates[:]:
109 # proc_date = proc_dates[55]
110     df_proc = df[(df['fullname_dt'] >= proc_date[0]) & (df['fullname_dt'] <
        proc_date[1])]
111     if len(df_proc) == 0 :
112         print("There is no data in {0} - {1} ...\n"\
113                 .format(proc_date[0].strftime('%Y%m%d'), proc_date[1].
        strftime('%Y%m%d')))
114
115     else :
116         print("df_proc: {}".format(df_proc))
117
118         #check file exist??
119         output_fullname = '{0}{1}_{2}_{3}_{4}_{5}_{6}_{7}_{8}_alldata_mean.nc'
         \
120                 .format(save_dir_name, DATAFIELD_NAME,
121                 proc_date[0].strftime('%Y%m%d'), proc_date[1].strftime('%Y
        %m%d'),
122                 str(Llon), str(Rlon), str(Slat), str(Nlat), str(resolution
        ))
123
124         if False and os.path.exists('{0}'.format(output_fullname)) :
125                 print('{0} is already exist...'.format(output_fullname))
126
127         else :
128             #if os.path.exists('{0}'.format(output_fullname)):
129             #    os.remove('{0}'.format(output_fullname))
130
131             print("Starting {0}\n".format(output_fullname))
132             output_fullname_el = output_fullname.split("/")
133             output_fileneme_el = output_fullname_el[-1].split("_")
134
135             alldata_3Ds = np.empty((0, int((Rlon-Llon)/resolution), int((Nlat-
        Slat)/resolution)))
136             for fullname in df_proc["fullname"] :
137                 #fullname = df_proc["fullname"][0]
138
139                 alldata = np.load(fullname, allow_pickle=True)
140
141                 if len(alldata.shape) == 3 :
142                     print("error")
```

```
143                     alldata = np.empty((int((Rlon-Llon)/resolution), int((Nlat
    -Slat)/resolution)))
144                 else :
145                     for i in range(alldata.shape[0]):
146                         for j in range(alldata.shape[1]):
147                             if len(alldata[i,j]) == 0 :
148                                 alldata[i,j] = np.nan
149                             else :
150                                 alldata[i,j] = np.mean(list(map(lambda x:x[1],
     alldata[i,j])))
151
152                 if alldata_3Ds.shape[0] == 0 :
153                     alldata_3Ds = alldata.reshape(1, alldata.shape[0], alldata
    .shape[1])
154                     print("alldata_3Ds.shape : True\n{}".format(alldata_3Ds.
    shape))
155                 else :
156                     alldata_3Ds = np.append(alldata_3Ds, alldata.reshape(1,
    alldata.shape[0], alldata.shape[1]), axis=0)
157                     print("alldata_3Ds.shape : Flase\n{}".format(alldata_3Ds.
    shape))
158
159             alldata_3Ds = alldata_3Ds.astype('float64')
160
161             print("alldata_3Ds.shape : final\n{}".format(alldata_3Ds.shape))
162             alldata = np.nanmean(alldata_3Ds, axis=0, keepdims=True)
163             print("alldata.shape :\n{}".format(alldata.shape))
164             print("alldata :\n{}".format(alldata))
165
166             alldata = alldata.reshape(alldata.shape[1], alldata.shape[2])
167             #alldata = alldata.transpose()
168             print("alldata.shape :\n{}".format(alldata.shape))
169             print("alldata :\n{}".format(alldata))
170             ds = nc.Dataset('{0}'.format(output_fullname), 'w', format='
    NETCDF4')
171
172             #time = ds.createDimension('time', filename_el[2])
173             time = ds.createDimension('time', None)
174
175             lon = ds.createDimension('longitude', alldata.shape[0])
176             lat = ds.createDimension('latitude', alldata.shape[1])
177             times = ds.createVariable('time', 'f4', ('time',))
178
179             lons = ds.createVariable('longitude', 'f4', ('longitude',))
180             lats = ds.createVariable('latitude', 'f4', ('latitude',))
181             SST = ds.createVariable('SST', 'f4', ('time', 'latitude', '
    longitude',))
```

```
            SST.units = 'degree'

            lons[:] = np.arange(Llon, Rlon+resolution, resolution)
            lats[:] = np.arange(Slat, Nlat+resolution, resolution)
            #lons[:] = np.arange(Llon, Rlon+resolution, resolution)
            #lats[:] = np.arange(Slat, Nlat+resolution, resolution)

            SST[0, :, :] = alldata.transpose()

            #print('var size after adding first data', value.shape)
            #xval = np.linspace(0.5, 5.0, alldata.shape[1]-1)
            #yval = np.linspace(0.5, 5.0, alldata.shape[0]-1)
            #value[1, :, :] = np.array(xval.reshape(-1, 1) + yval)

            ds.close()
```

V.4 4.draw_HIST_and_MAP_statistics_AVHRR_asc_SST_NCfile.py

```

from glob import glob
import os
import sys

from netCDF4 import Dataset as NetCDFFile
import MODIS_hdf_utilities

log_file = os.path.basename(__file__)[:-3]+".log"
err_log_file = os.path.basename(__file__)[:-3]+"_err.log"
print ("log_file: {}".format(log_file))
print ("err_log_file: {}".format(err_log_file))

arg_mode = True
arg_mode = False

if arg_mode == True :
    from sys import argv # input option
    print("argv: {}".format(argv))

    if len(argv) < 2 :
        print ("len(argv) < 2\nPlease input L3_perid and year \n ex) aaa.py
    daily")
        sys.exit()
    elif len(argv) > 2 :
        print ("len(argv) > 2\nPlease input L3_perid and year \n ex) aaa.py
    daily")
        sys.exit()
```

```
    elif argv[1] == 'daily' or argv[1] == 'weekly' or argv[1] == 'monthly' :
        L3_perid = argv[1]
        print("{} processing started...".format(argv[1]))
    else :
        print("Please input L3_perid \n ex) aaa.py daily")
        sys.exit()
else :
    L3_perid, resolution = "monthly", 0.5

# Set Datafield name
DATAFIELD_NAME = "AVHRR_SST"

#Set lon, lat, resolution
Llon, Rlon = 115, 145
Slat, Nlat = 20, 55

#set directory
base_dir_name = "../L3_{0}/{0}_{1}_{2}_{3}_{4}_{5}_{6}/".format(DATAFIELD_NAME
    , str(Llon), str(Rlon),
                                                        str(Slat), str(Nlat),
    str(resolution), L3_perid)
save_dir_name = base_dir_name
#save_dir_name = "../L3_{0}/{0}_{1}_{2}_{3}_{4}_{5}_{6}/".format(
    DATAFIELD_NAME, str(Llon), str(Rlon),
#                                                        str(Slat), str(Nlat),
     str(resolution), L3_perid)

#### make dataframe from file list
fullnames = sorted(glob(os.path.join(base_dir_name, '*mean.nc')))

print("len(fullnames): {}".format(len(fullnames)))

for fullname in fullnames :
    #fullname = fullnames[0]
    print("Starting {0}\n".format(fullname))
    fullname_el = fullname.split("/")
    filename_el = fullname_el[-1].split("_")
    #if os.path.exists('{0}_mean.npy'.format(fullname[:-4])) :
    #    print('{0}_mean.npy is already exist...'.format(fullname[:-4]))
    nc_data = NetCDFFile(fullname) # note this file is 2.5 degree, so low
    resolution data
    lat = nc_data.variables['latitude'][:]
    lon = nc_data.variables['longitude'][:]
    time = nc_data.variables['time'][:]
    SST = nc_data.variables['SST'][:] # SST
```

```
    if False and os.path.exists("{0}{1}_{2}_hist.pdf"\
        .format(base_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME)) :
        print("{0}{1}_{2}_hist.pdf is already exist..."\
              .format(save_dir_name, fullname_el[-1][:-4], DATAFIELD_NAME))
    else :
        try :
            plt_hist = MODIS_hdf_utilities.draw_histogram_SST_NC(SST, lon, lat
    , fullname, DATAFIELD_NAME)
            plt_hist.savefig('{0}_hist.pdf'.format(fullname[:-3]))
            print('{0}_hist.pdf is created...'.format(fullname[:-3]))
            plt_hist.close()
        except Exception as err :
            MODIS_hdf_utilities.write_log(err_log_file, err)
            continue

    if False and os.path.exists('{0}_map.png'.format(fullname[:-3])) :
        print('{0}_map.png is already exist'.format(fullname[:-3]))

    else :

        try :

            plt_map = MODIS_hdf_utilities.draw_map_SST_nc(SST, lon, lat,
    save_dir_name, fullname, DATAFIELD_NAME, Llon, Rlon, Slat, Nlat)

            plt_map.savefig('{0}_map.png'.format(fullname[:-3]))
            print('{0}_map.png is created...'.format(fullname[:-3]))
            plt_map.close()

        except Exception as err :
            MODIS_hdf_utilities.write_log(err_log_file, err)
            continue
```

References

[1] Wu, R., & Kirtman, B. P. (2007). Regimes of seasonal air–sea interaction and implications for performance of forced simulations. Climate dynamics, 29(4), 393–410.

[2] 정은실 (2019). 한반도에서 위험기상 발생 시 나타나는 해수면온도 변동의 특성. *한국지구과학회지*, 40(3), 240–258.

[3] Product definitions. `https://oceancolor.gsfc.nasa.gov/products/`. Accessed: 2021-06-30.

[4] Walton, C. C. (1988). Nonlinear multichannel algorithms for estimating sea surface temperature with avhrr satellite data. Journal of Applied Meteorology and Climatology, 27(2), 115–124.